НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 «Програмна інженерія»

на тему: «Веб-додаток для спільного управління фінансами громади»

Виконала:
студентка IV курсу, групи КП-52
Ісаєва Надія Михайлівна _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н,
Рибачок Н.А. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ЕІ ФЕЛ, к.т.н.,
Вунтесмері Ю.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018 р.

ЗАВДАННЯ
на дипломний проект студентці

Ісаєвій Надії Михайлівні

1. Тема проекту «Веб-додаток для спільного управління фінансами громади», керівник проекту Рибачок Наталія Антонівна, к.т.н., старший викладач, затверджені наказом по університету від «22» травня 2019 р. №1331-С

2. Термін подання студентом проекту «20» червня 2019 р.

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд існуючих програмних рішень;
   - обґрунтування вибору засобів реалізації ;
   - характеристика розробленої системи;
   - аналіз розробленої системи.

5. Перелік обов'язкового графічного матеріалу:
   - діаграма прецедентів (креслення);
   - структура бази даних (креслення);
   - структурна схема системи (плакат);
   - процеси перетворення даних (плакат).

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Ознайомлення з предметною областю	10.11.2018	
2.	Розроблення та узгодження технічного завдання	10.12.2018	
3.	Аналіз існуючих аналогів та розробка вимог	15.01.2019	
4.	Проектування архітектури додатку	05.02.2019	
5.	Аналіз та вибір засобів реалізації	17.02.2019	
6.	Створення структури бази даних	10.03.2019	
7.	Розробка інтерфейсу	22.03.2019	
8.	Програмна реалізація додатку	17.04.2019	
9.	Тестування системи	25.04.2019	
10.	Підготовка матеріалів першого розділу дипломного проекту	04.05.2019	
11.	Підготовка матеріалів другого розділу дипломного проекту	10.05.2019	
12.	Підготовка матеріалів третього розділу дипломного проекту	15.05.2019	
13.	Підготовка матеріалів четвертого розділу дипломного проекту	20.05.2019	
14.	Оформлення документації дипломного проекту	25.05.2019	

Студентка Н.М. Ісаєва

Керівник проекту Н.А. Рибачок

АНОТАЦІЯ

Даний дипломний проект присвячено створенню веб-додатку для спільного управління фінансами громади. Вибір теми проекту обґрунтований потребами громад в узгодженості цілей фінансування під час розподілу та управління коштами. Проект покликаний підвищити ефективність звітування та координації проектної діяльності колективу.

У роботі було проаналізовано потреби громад навчальних закладів щодо фінансової діяльності, визначено критерії оцінювання програмних продуктів для спільного управління коштами. Визначено переваги та недоліки аналогів, рівень їх відповідності потребам користувачів. Було сформовано та документовано вимоги до розроблюваного програмного забезпечення.

Проект реалізований у формі веб-додатку для громад навчальних закладів початкової та середньої освіти та може бути масштабований для використання іншими видами спільнот. Для початку роботи з додатком необхідна авторизація. Для зареєстрованих користувачів – керівників громад реалізовано функції управління фінансовими проектами та витратами. Учасникам громад надано функції вкладення та висування потенційних проектів. Всім зареєстрованим користувачам забезпечено доступ до перегляду балансу, історії внесків і генерування звітів.

У результаті роботи над дипломним проектом розроблено архітектуру веб-додатку, структуру бази даних та фінансового проекту колективу, а також дизайн інтерфейсу динамічних веб-сторінок.

ABSTRACT

This diploma project is devoted to the development of a web-based application for joint management of community finances. The choice of project theme is justified by the needs of different communities for coherence of funding objectives during the allocation and management of funds. Project is aimed to contribute to the efficiency of reporting and project activity coordination.

This work contains analysis of educational institutions community needs regarding financial activities. Criteria for evaluating software products for joint management of funds were defined. Advantages and disadvantages of similar software tools were determined and level of their correspondence to the users' needs was established. Software requirements for developed web application were formed and documented.

The project is implemented in the form of a web application for communities of primary and secondary education institutions and can be scaled for the use by other community types. To get started with the application use, authorization is required. Functions for managing financial projects and costs were implemented for community leaders registered users. Community members have been given functions for making investments and nominating potential projects.

Architecture of web application, structure of database and financial group project as well as interface design of dynamic web pages were created as a result of this diploma project.

АННОТАЦИЯ

Данный дипломный проект посвящен созданию веб-приложения для совместного управления финансами общины. Выбор темы проекта обоснован потребностями общин в согласованности целей финансирования при распределении и управлении денежными средствами. Проект призван повысить эффективность отчетности и координации проектной деятельности коллектива.

В работе были проанализированы потребности общин учебных заведений относительно финансовой деятельности, определены критерии оценки программных продуктов для совместного управления средствами. Определено преимущества и недостатки аналогов, уровень их соответствия потребностям пользователей. Требования к разрабатываемому программному обеспечению были сформированы и документированы.

Проект реализован в форме веб-приложения для общин учебных заведений начального и среднего образования и может быть масштабирован для использования другими видами сообществ. Для начала работы с приложением требуется авторизация. Обеспечено разделение ролей пользователей. Для зарегистрированных пользователей – руководителей реализованы функции управления финансовыми проектами и расходами. Участникам предоставлены функции вложения и выдвижения потенциальных проектов. Всем зарегистрированным пользователям обеспечен доступ к просмотру баланса, истории взносов и генерированию отчетов.

В результате работы над дипломным проектом разработана архитектура веб-приложения, структура базы данных и финансового проекта коллектива, а также дизайн интерфейса динамических веб-страниц.

ДП.045440-01-90 Веб-додаток для спільного управління фінансами громади.
Відомість проекту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проекту		
ДП.045440-02-91	Веб-додаток для	5	
	спільного управління		
	фінансами громади.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для	72	
	спільного управління		
	фінансами громади.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для	5	
	спільного управління		
	фінансами громади.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для	10	
	спільного управління		
	фінансами громади.		
	Керівництво користувача		
ДП.045440-06-99	Веб-додаток для	1	
	спільного управління		
	фінансами громади.		
	Концептуальний опис		
	поведінки системи.		
	Діаграма прецедентів		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Веб-додаток для	1	
	спільного управління		
	фінансами громади.		
	Структура бази даних.		
	ERD діаграма		
ДП.045440-08-98	Веб-додаток для	1	
	спільного управління		
	фінансами громади.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ВЕБ-ДОДАТОК ДЛЯ СПІЛЬНОГО УПРАВЛІННЯ ФІНАНСАМИ ГРОМАДИ

Технічне завдання

ДП.045490-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Н.А. Рибачок

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Н.М. Ісаєва

2018

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-додаток для спільного управління фінансами громади.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка у формі веб-додатку призначена для використання в якості засобу спільного управління коштами в громадах навчальних закладів з метою узгодження колективного фінансування, проектної діяльності та звітування.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1) керування громадою: створення, редагування, запрошення учасників, перегляд балансу та історії внесків;
2) керування проектами: створення, редагування, підтвердження, закриття, перегляд балансу та історії внесків;
3) керування внесками: підтвердження грошових внесків учасників громади;
4) керування витратами: списання коштів із балансу громади або проекту;

5) можливість приєднання до громади, перегляду фінансових даних громади та особистої сторінки учасника, генерування звітів;
6) можливість пропонування та коментування проектів.

Розробку виконати на платформі Flask.

Додаткові вимоги:

1) розділення відповідальності на декілька ролей користувачів;
2) наявність адаптивного користувацького інтерфейсу з використанням традиційних елементів;
3) надання доступу до розгорнутої інформації про фінансові проекти;
4) забезпечення надійності та безпеки використання системи.

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Концептуальний опис поведінки системи. Діаграма прецедентів»;
   - «Структура бази даних. ERD діаграма».

6. ЕТАПИ ПРОЕКТУВАННЯ

Ознайомлення з предметною областю 10.11.2018

Розроблення та узгодження технічного завдання 10.12.2018

Аналіз існуючих аналогів та розробка вимог 15.01.2019

Проектування архітектури додатку 05.02.2019

Аналіз та вибір засобів реалізації 17.02.2019

Створення структури бази даних 10.03.2019

Розробка інтерфейсу 22.03.2019

Програмна реалізація додатку 17.04.2019

Тестування системи 25.04.2019

Підготовка матеріалів першого розділу 04.05.2019

Підготовка матеріалів другого розділу 10.05.2019

Підготовка матеріалів третього розділу 15.05.2019

Підготовка матеріалів четвертого розділу 20.05.2019

Оформлення документації дипломного проекту 25.05.2019

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ СПІЛЬНОГО УПРАВЛІННЯ ФІНАНСАМИ ГРОМАДИ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Н.А. Рибачок

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Н.М. Ісаєва

2019

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Автентифікація – перевірка достовірності пред'явленого користувачем ідентифікатора.

Авторизація – надання прав на виконання дій в системі та процес перевірки прав при спробі виконання цих дій.

Бюджет – грошове вираження збалансованості доходів та витрат громади за певний період.

Веб-браузер – програмне забезпечення для під'єднаного до мережі Інтернет пристрою, що дає можливість взаємодії з даними на гіпертекстовій веб-сторінці.

Веб-фреймворк – каркас, призначений для створення динамічних веб-додатків та сервісів, що спрощує розробку та зменшує дублювання коду завдяки наявності готових компонентів.

Громада – група людей, об'єднаних спільністю становища, інтересів, що ставить перед собою певні спільні завдання, має спільні прагнення, цілі та структури.

Маршрутизація URL – встановлення відповідності між адресою ресурсу та функцією-обробником, що викликається при запиті ресурсу.

Модальне вікно – діалогове вікно, що блокує роботу користувача з додатком, доки користувач не закриє вікно після введення або отримання певної інформації.

Мультипарадигмальна мова програмування – мова програмування з підтримкою кількох стилів написання та організації структури програмного забезпечення.

Рендеринг – конвертація шаблону в готову HTML-сторінку.

Фінансові ресурси – кошти, що перебувають у розпорядженні громади та призначені для виконання керівництвом громади певних фінансових зобов’язань.

Шаблонізатор – ПЗ, що дозволяє використовувати HTML-шаблони для генерації кінцевих HTML-сторінок.

Envelope Budgeting System – система організації бюджету, побудована на його розподілі на категорії та фіксуванні доходів/витрат за кожною окремою категорією.

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

СУБД – система управління базами даних.

ACID – Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізольованість, довговічність);

AJAX – Asynchronous JavaScript and XML (асинхронний JavaScript та XML);

API – Application Programming Interface (прикладний програмний інтерфейс);

CSRF – Cross Site Request Forgery (міжсайтова підробка запиту);

CSS – Cascading Style Sheets (каскадна таблиця стилів);

DOM – Document Object Model (об’єктна модель документа);

DRY – Don't repeat yourself (не повторюй себе);

ERD – Entity Relationship Diagram (діаграма «сутність-зв’язок»);

GUI – Graphical User Interface (графічний інтерфейс користувача);

HTML – HyperText Markup Language (мова розмітки гіпертексту);

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту);

JSON – JavaScript Object Notation (запис об'єктів JavaScript);

MVC – Model-View-Controller (модель-подання-контролер);

ORM – Object-Relation Mapping (об’єктно-реляційне відображення);

PyPI – Python Package Index (каталог пакетів Python);

REST – Representational State Transfer (передача стану подання);

RWD – Responsive web design (адаптивний веб-дизайн);

SMTP – Simple Mail Transfer Protocol (простий протокол пересилання пошти);

UML – Unified Modeling Language (уніфікована мова моделювання);

URL – Uniform Resource Locator (уніфікована адреса ресурсу);

WSGI – Web Server Gateway Interface (інтерфейс шлюзу веб-сервера).

ВСТУП

Для ефективної діяльності колективу вкрай важливими є взаємодія його учасників між собою та з керівником колективу. Одними з найбільш проблематичних аспектів взаємодії у громадах вважаються управління фінансами та проектами. Навіть в невеликих колективах виникають проблеми залучення всіх учасників до формування спільного бюджету та інформування їх про розподіл коштів. Ці проблеми потребують рішень особливо гостро у громадах із суспільно важливими цілями, зокрема навчання та виховання учнів у закладах початкової та середньої освіти. В навчальних закладах потрібні інструменти забезпечення прозорого спільного управління фінансовими ресурсами для постійного поліпшення матеріальної бази закладу та підтримки процесу виховання дітей.

За розподіл та управління колективними фінансами вказаних вище громад зазвичай несе відповідальність батьківський комітет, а формується фінансовий ресурс навчальним закладом та приватними особами, а саме батьками учнів закладу [1]. Налагодження каналів комунікації є надзвичайно важливим для організації та проведення спільної фінансової діяльності: при прийнятті спільних рішень про витрати, для формування та надання звітності про зібрані та витрачені кошти, для можливості висування власних ідей для спільного фінансування в майбутньому.

На жаль, на практиці поширені прецеденти відсутності належного звітування перед колективом, непорозуміння між колективом та керівником щодо цілей витрачених коштів, неефективне використання часу на зустрічі для координації діяльності та погіршення відносин в колективі через можливі заборгованості учасників.

Виникає необхідність створення автоматизованої системи для спільного управління фінансами громади, яка б вирішувала вищезгадані проблеми для різного розміру колективів. Тому пропонується розробка програмного засобу з функціями відслідковування вкладених та витрачених

на спільні проекти коштів, генерації фінансової звітності громади та узгодження цілей фінансування.

Отже, даний дипломний проект присвячено актуальній задачі – створенню веб-додатку для спільного управління фінансами громади.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

Перед пошуком існуючих рішень було вирішено зібрати основні потреби громад навчальних закладів початкової та середньої освіти щодо управління спільними фінансами. Зокрема, до автоматизованої системи було висунуто такі потреби:

– доступ до розгорнутої інформації про фінансові проекти;

– можливість висунення власних проектів;

– можливість обговорення проектів;

– розділення прав доступу;

– підтримка всіх ОС та пристроїв;

– невисокий поріг входження для ефективного використання ПЗ.

1.1. Обґрунтування вибору критеріїв оцінювання програмних засобів

В процесі аналізу вимог до системи було виділено наступні функціональні вимоги:

1. Необхідність авторизації для роботи з додатком та розділення відповідальності на декілька ролей користувачів: Адміністратор (для керівника громади) та Учасник громади.
2. Можливість створення і редагування Адміністратором нової громади та додавання одного або кількох Учасників до створеної громади.
3. Адміністратор має доступ до переліку Учасників та можливість видалення Учасника із громади.
4. Адміністратор має можливість створювати, редагувати та видаляти проекти, що потребують фінансування громади, розпочинати фінансування проекту, здійснювати операцію «Витрати» на проект із загального балансу громади та закривати проект після здійснення

всіх необхідних витрат. Також Адміністратор підтверджує здійснені внески Учасників громади.

5. Адміністратор має доступ до перегляду поточного балансу громади, історії внесків Учасників громади та можливість генерування звітів за внесками і проектами громади.
6. Забезпечення можливості Учасника приєднатися до громади за посиланням.
7. Наявність особистої сторінки Учасника з даними про власні вкладення, фінансовані проекти, борги за наявності, строки проектів, що потребують фінансування.
8. Можливість Учасника коментувати наявні проекти, висувати власні проекти на фінансування, переглядати поточний баланс громади (обсяг внесених та витрачених коштів), історію внесків всіх учасників громади, генерування звітів за внесками і проектами громади.
9. Учасник може здійснювати операцію «Вкладення» до балансу громади за себе та за іншого учасника громади.

Також було визначено нефункціональні вимоги:

1. Підтримка роботи додатку в найбільш поширених операційних системах (Linux, Windows, Mac OS). Перевагою може бути наявність мобільної версії додатку для користувачів Android та iPhone.
2. Простота та зрозумілість інтерфейсу додатку, невисокий поріг входу та мінімальна кількість дій користувача для ефективної роботи з додатком.
3. Наявність україномовної локалізації.
4. Детальний опис структури фінансованого проекту: назва, категорія, кошторис, строки внесків та строки реалізації. Вимога випливає із необхідності генерування звітів за проектами громади.

1.2. Аналіз існуючих програмних рішень

1.2.1. AlzexFinance

AlzexFinance – програмне забезпечення для ведення бухгалтерії та планування спільного бюджету. Головна ідея програми полягає у розділенні доходів та витрат на категорії, що дозволяє користувачам контролювати куди і в якій кількості витрачаються кошти. Система категорій є деревовидною з необмеженою вкладеністю. Доходи та витрати можна розділяти за учасниками колективу та привласнювати їм спеціальні мітки, відстежувати зроблені іншими записи, захищати свої записи від змін та приховувати їх від інших учасників.

Одним із ключових понять у програмі є транзакція – будь-який рух коштів (дохід, витрата чи переказ коштів). Транзакція завжди зв’язана з рахунком користувача. Транзакції групуються за категоріями, учасниками, проектами та описом.

Для планування спільного бюджету наявний інструмент «Календар» зі зручним представленням усіх платежів на будь-який день. За замовчуванням програма для ведення бухгалтерії показує один місяць, але користувач може обрати інший період часу. Рішенням для складання спільного бюджету є «Планувальник», що дозволяє автоматично створювати регулярні транзакції.

Розділ «Бюджет» містить інструменти контролю за прогресом у досягненні фінансових цілей. Основою контролювання бюджету в програмі є вимога меншого загального обсягу витрат порівняно із загальним обсягом доходу. Користувач має змогу налаштувати бюджет, і програма буде автоматично рахувати доходи і витрати та порівнювати їх із запланованими сумами витрат на період тижня, місяця або року. Програма надає графічне зображення поточного стану бюджету у вигляді прогрес-бару, що показує обсяг коштів, які можна витратити без збитку, або наскільки було перевищено плановане обмеження.

Програма містить функції генерування звітів для аналізу спільного бюджету в числовому та графічному вигляді. Є можливість переглянути підсумки за обраний період з урахуванням поділу на категорії та учасників.

Основна функціональність програми поставляється безкоштовно та доступна користувачам Android, iPhone та Windows, але ведення єдиного обліку кількома учасниками надається лише у преміум-версії програми, яка є платною. Крім цього, недоліками програми є відсутність української локалізації та перевантажений інтерфейс користувача. Дане програмне забезпечення вимагає від користувача часу для розуміння та ефективного використання функцій; частково відповідає критеріям функціональних вимог 1, 4, 5, 7, 8, 9 та 1, 4 критеріям нефункціональних вимог.

1.2.2. Team Money

Team Money – програмне забезпечення, доступне у вигляді мобільного додатку користувачам Android для управління коштами, що належать команді. Функціональність додатку забезпечує всіх членів команди можливістю переглядати інформацію про вкладення: хто зробив вкладення, в якому обсязі, на що використовуються внесені кошти, а також дозволяє перераховувати кошти між членами команди. Учасники також можуть здійснювати пошук транзакцій за датою, описом, тегами та іншими атрибутами.

Додаток передбачає три типи транзакцій: Capital, Transfer та Expense. Capital – це обсяг коштів, який має команда на поточний момент часу; транзакція потребує здійснення на початку роботи з додатком. Сума коштів, внесених за транзакцією Capital, реєструється на балансі користувача, який здійснив транзакцію. В процесі роботи з додатком користувачі можуть переказати кошти на баланс інших учасників (Transfer) та витратити кошти (Expense). Після отримання переказу учасник може витратити кошти або здійснити переказ на інший баланс.

Програмний засіб надає звіти за транзакціями, в яких користувач може переглянути поточний обсяг коштів на загальному та власному балансі, перелік транзакцій, класифікацію витрат за тегами. Звіти можуть бути завантажені на пристрій користувача у форматі CSV.

Додаток дозволяє участь одного користувача у кількох командах, редагування та коментування іншими користувачами здійснених транзакцій, прикріплення файлів до транзакцій. Наявна функціональність для додавання учасників до команди за номером мобільного телефону. Серед переваг додатку варто зазначити простий для розуміння інтерфейс, можливість використання без підключення до мережі Інтернет та використання в якості додатку для формування списку задач. Недоліками є відсутність української локалізації та функцій пропонування проектів для майбутнього фінансування, а також неможливість здійснити транзакцію Expense за іншого учасника. Використання продукту є безкоштовним. Таким чином, додаток відповідає 1, 2, 3, 5, 9 критеріям функціональних вимог та частково 6; 2 критерію нефункціональних вимог.

1.2.3. Goodbudget

Goodbudget – це менеджер по роботі з коштами і трекер витрат, призначений для планування колективного бюджету, постановки фінансових цілей та контролю витрат в режимі реального часу. Надає широкий набір можливостей користувачам Android та iPhone, серед яких аналіз звітів та контроль витрат, завантаження звітів у форматі CSV, здійснення транзакцій за розкладом, розподілення витрат, пошук транзакцій, додавання та редагування учасників, а також якісний UX – мінімізована кількість дій користувача для ефективної роботи з додатком. Програмний засіб приваблює зручним та простим у використанні інтерфейсом, відсутністю реклами, можливістю збереження історії транзакцій до семи років; може використовуватись в якості книги бухгалтерського обліку, для побутового планування бюджету та особистих

цілей. Функції додатку випливають із Envelope-організації бюджету, тобто розподілу бюджету на витрати за категоріями. Доступна генерація звітів як за категоріями витрат, так і по загальному обсягу транзакцій. Недоліками додатку є необхідність придбання платної підписки для додавання учасників бюджету та необмеженої кількості категорій, а також наявність лише англійської локалізації. Отже, додаток відповідає критеріям функціональних вимог 1, 2, 3, 5 частково 4, 6, 7, 8, 9 та критеріям нефункціональних вимог 2, 4.

1.2.4. Acasa – Manage bills

Acasa – програмний засіб, покликаний вирішувати задачу колективного управління фінансами в різноманітних галузях, заощаджуючи час, зусилля та потенційні незручності щодо платежів. Розробники пропонують мобільний додаток Acasa для випадків контролю сімейного бюджету, розподілу фінансів тимчасового проекту, оплати рахунків спільного житла тощо. Пропонуються функції для відстеження та розподілу господарчих витрат, перекази коштів до балансу інших учасників колективу, сповіщення про дії інших учасників. Головна сторінка додатку інформує про майбутні витрати, прострочені залишки та дозволяє кожному бачити, якщо хтось із учасників не здійснив платіж. Додаток буде зручним та зрозумілим рішенням для контролю регулярних внесків при спільному веденні господарства. Але функціональність для фінансового контролю проектної діяльності громади є вкрай обмеженою. Програмний засіб відповідає критеріям функціональних вимог 1, 2, 3, 6, 9 та 2 критерію нефункціональних вимог.

1.3. Результати аналізу

За результатами огляду існуючих рішень проблеми управління спільними фінансами можна відзначити, що жодне з них цілком не відповідає поставленим вимогам. В табл. 1 узагальнено відповідність

проаналізованих програмних рішень критеріям оцінювання автоматизованої системи з точки зору функціональних вимог.

Таблиця 1

Порівняльна характеристика існуючих рішень

	1	2	3	4	5	6	7	8	9
AlzexFinance	+			+	+		+\–	+\–	+\–
Team Money	+	+	+		+	+\–			+
Goodbudget	+	+	+\–	+\–	+	+\–	+\–	+\–	+\–
Acasa	+	+	+			+			+

Як показує таблиця, програмний продукт Acasa відповідає невеликій кількості поставлених вимог, що очевидно з факту його основного призначення для ведення господарчого бюджету. Сервіси AlzexFinance та Team Money надають функціональність для спільного управління фінансами та звітності, проте є відносно обмеженими щодо проектної діяльності. Оптимальним вибором між двома сервісами є AlzexFinance з хоча й складним інтерфейсом, проте великим набором інструментів планування бюджету.

Найбільшу кількість функціональних переваг надає програмний засіб Goodbudget. На жаль, додаток має лише англійську локалізацію та передбачає платне використання для груп користувачів, що є основною вимогою до системи управління спільними фінансами громад.

З точки зору нефункціональних вимог жоден з додатків не забезпечує вичерпної структури звітів за профінансованими проектами, хоча AlzexFinance є досить інформативним.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Обґрунтування вибору типу програмного забезпечення

Для розроблення автоматизованої системи спільного управління фінансами громади відповідно до вимог, що випливають з постановки задачі, необхідні:

– серверна частина системи;

– клієнтська частина системи;

– доступ до бази даних для збереження результатів діяльності громади.

Було проаналізовано кілька варіантів розробки програмного продукту для вирішення проблем користувачів: розробка веб-додатку, десктопного додатку та мобільного додатку [2]. Із врахуванням загальних переваг та недоліків даних способів, а також результатів першого розділу дипломного проекту з дослідженням існуючих рішень, серед яких представлені відповідним чином розроблені продукти, наводиться табл. 2 – узагальнення трьох варіантів.

Таблиця 2

Порівняльна характеристика способів розробки

Ознака	Веб-додаток	Десктопний додаток	Мобільний додаток
Вимоги до обладнання	Відсутність апаратних вимог до клієнта, крім сучасного веб-браузера, оскільки обчислення виконуються сервером	Розроблення для конкретної цільової ОС; обчислювальні навантаження потребуватимуть відповідної потужності апаратної частини клієнта	Розроблення для конкретної цільової платформи; повна сумісність із апаратним забезпеченням пристрою

Продовження табл. 2

Встановлення та обслуговування	Не вимагає встановлення та обслуговування	Вимагає встановлення та обслуговування	Вимагає встановлення та обслуговування
Збір даних	Дані клієнта завантажуються в будь-якому разі, зберігання може бути автоматичним	Збереження даних користувача можливе після завантаження додатку	Збереження даних користувача можливе після завантаження додатку
Розділення прав доступу	Легке і звичне забезпечення розділення прав користувачів	Складно забезпечити модель даних користувачів з привілеями для певних користувачів	Можливе забезпечення розділення прав користувачів
Вартість розробки	Розробка дешевша, ніж розробка десктопного додатку для будь-якої цільової платформи	Необхідні витрати на розробку для кожної цільової платформи	Кожна мобільна платформа потребує витрат, відповідної мови програмування та засобів розробки

Зважаючи на порівняльний аналіз, пропонується розробка системи у вигляді веб-додатку для спільного управління фінансами громади. Незалежність від цільової платформи, легкість встановлення, підтримки та обслуговування, можливість доступу користувачів із будь-якого пристрою з підключенням до мережі Інтернет стали факторами вибору даного способу. Розробка веб-додатку також надає ширші можливості в реалізації інтерфейсу користувача та забезпеченні спільного доступу для групи користувачів.

2.2. Вибір мови програмування для розробки серверної частини веб-додатку

Для реалізації серверної частини веб-додатку було вирішено обрати мультипарадигмальну мову програмування, з підтримкою ООП, великим об’ємом стандартної бібліотеки та наявністю вибору серед множини веб-фреймворків. Також до мови програмування була висунута вимога інтерпретованості з метою підвищення швидкості розробки та налагодження коду розроблюваної системи.

2.2.1. Мова PHP

PHP – мова програмування загального призначення, яка створена переважно для веб-розробки. Код PHP може виконуватися як за допомогою інтерфейсу командного рядка, вбудованого в HTML-код, так і в поєднанні з різними системами веб-шаблонів, системами керування веб-контентом і веб-фреймворками. Це найпопулярніше рішення для веб-розробки з точки зору кількості веб-сайтів (серед яких Facebook та Вікіпедія), де використовується ця мова, та систем керування контентом, побудованих на її основі (Wordpress, Drupal та Bolt серед найбільш відомих) [3].

Мова PHP знайшла застосування і за межами контексту веб-розробки, з її допомогою розробляються:

– GUI-додатки;

– кросплатформні додатки;

– програмні системи для управління дронами [4].

Мова є інтерпретованою; код обробляється інтерпретатором PHP, що зазвичай реалізований модулем у веб-сервері. Інтерпретованість дозволяє достатньо швидко обробляти написані сценарії, в порівнянні з іншими мовами (наприклад, Perl).

Веб-сервер комбінує результати інтерпретованого та виконаного PHP-коду, який може містити будь-які типи даних, з-поміж них зображення, зі згенерованою веб-сторінкою, що передається на сторону клієнта. На відміну

від скриптових мов програмування, користувач не має доступу до PHP-коду, тому що браузер отримує згенерований HTML-код в готовому вигляді.

Стандартний інтерпретатор мови PHP вільно розповсюджується, може бути розгорнутий на більшості веб-серверів практично на будь-якій операційній системі та платформі безкоштовно.

Синтаксис PHP подібний до синтаксису мови C, містить запозичені конструкції з Perl (наприклад, асоціативні масиви). В процесі історичного розвитку мова розширювалась та доповнювалась новими і запозиченими з інших мов можливостями без встановлення послідовних правил та конвенцій. З одного боку, це спричинило неузгодженість синтаксису PHP, але з іншого боку, мова отримала невисокий поріг входження для програмістів з різних галузей, обумовлений наявністю традиційних та знайомих конструкцій.

PHP є мультипарадигмальною мовою програмування, з підтримкою функціональної, процедурної та об'єктно-орієнтованої парадигм [5]. Починаючи лише з п’ятої версії мова підтримує ООП: три основні механізми (інкапсуляцію, наслідування та поліморфізм), інтерфейси, абстрактні класи та методи. Множинне наслідування не підтримується.

Інтерпретатор PHP забезпечує обробку скриптів зі збереженням кросплатформності розроблюваних додатків. Швидкодія додатків може бути збільшена з допомогою спеціального програмного забезпечення - акселераторів.

Програміст не має піклуватися про розподіл і звільнення пам'яті, тому що ядро PHP надає засоби для автоматичного керування пам'яттю; вся виділена пам'ять повертається в систему після завершення роботи скрипта.

Стандартна бібліотека мови PHP являє собою колекцію класів та інтерфейсів для вирішення типових проблем PHP. Бібліотека доступна розробнику за замовчуванням та загалом містить класи-ітератори для ітерацій по каталогу, масиву, дереву XML.

Найбільш поширеними PHP-фреймворками для веб-розробки є Laravel, CodeIgniter, Symfony, Zend.

2.2.2. Мова JavaScript

JavaScript – високорівнева інтерпретована мультипарадигмальна мова програмування з підтримкою об’єктно-орієнтованого, функціонального та імперативного стилів; реалізація стандарту ECMAScript. Широке застосування мова JavaScript знайшла в галузі веб-розробки, створенні сценаріїв веб-сторінок. Мова надає можливості для реалізації як клієнтської, так і серверної частин веб-додатків, взаємодії з користувачем на стороні клієнта, керування браузером, асинхронного обміну даними з сервером, динамічної зміни структури та зовнішнього вигляду сторінок. JavaScript має С-подібний синтаксис, динамічну типізацію, автоматичне управління пам’яттю, прототипно-орієнтований підхід до реалізації ООП [6, 7].

Типові випадки використання мови JavaScript:

– створення сценаріїв для інтерактивних веб-сторінок;
– розробка односторінкових веб-додатків (клієнтські фреймворки React, AngularJS, Vue.js);
– розробка серверної частини веб-додатків (фреймворк Node.js);
– розробка кросплатформних додатків (фреймворки Electron, NW.js);
– розробка нативних мобільних додатків (React Native, Cordova);
– написання сценаріїв для прикладного ПЗ.

Мова не є об’єктно-орієнтованою в класичному розумінні через використання прототипного програмування, в якому поняття класу відсутнє, а наслідування відбувається шляхом клонування прототипу – існуючого примірника об’єкту.

JavaScript надає властивості, запозичені з функціональних мов програмування, зокрема функції як об’єкти першого класу, об’єкти як списки, каррінг, анонімні функції, замикання, що надає мові додаткової гнучкості, втім не є значною перевагою в контексті розроблюваної системи.

Мова використовується в AJAX, поширеному підході до побудови інтерактивних користувацьких інтерфейсів веб-додатків, що полягає в «фоновому» асинхронному обміні даними браузера з веб-сервером з метою пришвидшення їх взаємодії [8].

Варто відзначати відсутність у складі мови наступних складових:

– стандартна бібліотека (відсутній інтерфейс для роботи з файловою системою, управління потоками введення-виведення);
– стандартні інтерфейси до веб-серверів та баз даних;
– система управління пакетами для відслідковування та автоматичного встановлення залежностей.

Для розробки серверної частини додатків мова не може застосована безпосередньо, потрібне використання спеціальних JavaScript-рушіїв [9]:

– Rhino – перетворює JavaScript-скрипти в Java класи, може працювати в інтерпретованому режимі, не має вбудованої підтримки для об’єктів браузера;
– SpiderMonkey – рушій вбудовується в інші застосунки, які надають робоче оточення для JavaScript; містить компілятор, інтерпретатор, декомпілятор, прибиральник сміття і стандартні класи;
– V8 – найбільш швидкий та потужний рушій, в якому JavaScript-код безпосередньо перетворюється в асемблер цільового процесора, що дозволяє обійти за швидкістю інші засоби.

На основі V8 побудована популярна серверна платформа Node.JS. Саме впровадження Node.JS перетворило мову JavaScript на мову загального використання з можливістю виконувати JavaScript-скрипти на сервері та відправляти користувачам результати їх виконання.

Окрім рушія, Node.JS постачає вбудований сервер та базовий набір бібліотек, а також надає можливість асинхронної роботи з файлами та мережевими пристроями.

Наразі мова JavaScript є однією з найпопулярніших в інтернеті, підтримує відповідність сучасним стандартам веб-програмування та надає

засоби для серверної розробки, проте не всі її інструменти є достатньо зрілими та протестованими і не завжди мають вичерпну документацію [10]. Також асинхронність самої мови в поєднанні з однопоточністю платформи для серверної розробки сприяє зниженню обчислювальної продуктивності.

2.2.3. Мова Ruby

Ruby – інтерпретована мова програмування з повною підтримкою ООП та динамічною типізацією; якісно вирізняється високою ефективністю розробки програм. Синтаксис Ruby подібний до синтаксису мови Perl, а підхід до реалізації ООП нагадує відповідний підхід мови програмування Smalltalk; деякі властивості були запозичені з інших мов – Java, Python, Lisp [11].

Інтерпретатор мови розповсюджується безкоштовно та доступний для більшості платформ і ОС. Незалежно від ОС мова надає власну реалізацію багатопоточності та автоматичну збірку сміття.

Ruby має широкий спектр застосування – з допомогою мови:

– розробляються програмні продукти різного призначення;

– відбувається автоматизація та налаштування додатків;

– можливе написання адміністративних утиліт.

Динамічні засоби мови з одного боку сприяють ефективності програмування, однак знижують продуктивність.

Ruby не містить примітивні типи, надаючи перевагу цілком об’єктно-орієнтованій організації: всі дані у програмах є об’єктами, всі функції – методами. Мова не має підтримки множинного наслідування.

Мультипарадигмальність втілена в наявності рис процедурного стилю (визначення функцій та змінних поза межами класу), об’єктно-орієнтованого та функціонального стилю (анонімні функції, замикання); також розробнику доступні рефлексія, метапрограмування, інформація про типи змінних на стадії виконання.

Мова постачається зі стандартною бібліотекою великого об’єму, що містить засоби для роботи з мережевими протоколами на клієнтській та серверній стороні, засоби для обробки та представлення різноманітних форматів даних.

Додаткові можливості надають бібліотеки Ruby для модульного тестування, роботи з архівами, датами, матрицями та засоби для системного адміністрування, розподілених обчислень. Бібліотека Ruby reports призначена для легкої реалізації звітів і створення діаграм на основі даних з БД та текстових файлів CSV.

Мова Ruby є порівняно молодою, незадовго після її створення з’явився веб-фреймворк Ruby on Rails, що користується популярністю і сьогодні через легкість побудови типових веб-додатків. Також існують веб-фреймворки Nitro, Webby, Sinatra.

2.2.4. Мова Python

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією; підтримує кілька парадигм програмування: об'єктно-орієнтовану, процедурну, функціональну та аспектно-орієнтовану [12].

Мова використовується для вирішення різноманітних проблем в галузі розробки ПЗ:

– створення десктопних додатків з GUI (ігри, наукові програми);

– створення веб-додатків та веб-фреймворків;

– розробка програм для підприємств та бізнесу;

– розробка ОС;

– прототипування.

Інтерпретатор Python та стандартні бібліотеки мови доступні як у скомпільованій, так і в вихідній формі для всіх основних платформ та ОС. Стандартна бібліотека містить великий об’єм корисних функцій. Із

мінімалізмом синтаксису мова орієнтована на підвищення продуктивності програміста та читабельності коду.

Python є мовою програмування загального призначення, яка продовжує активно розвиватися; до мови додаються та змінюються існуючі властивості, проте основні риси залишаються сталими. Серед таких зокрема динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізми обробки виключень, підтримка багатопоточності, розвинені високорівневі структури даних та динамічна перевірка типів.

Портованість мови дозволяє працювати на будь-якій з відомих платформ; втім, на відміну від інших портованих систем, Python надає підтримку характерних для платформи технологій. Існує спеціальна версія мови для віртуальної машини Java – Jython, рішення для інтеграції з платформою Microsoft .NET – IronPython, Python.Net.

Всі дані в програмах, написаних на Python, є об'єктами, в тому числі функції, методи, класи та модулі.

Мова містить як примітивні типи, так і вбудовані колекції - списки, кортежі, словники, множини.

Система класів Python підтримує одиночне та множинне наслідування, метапрограмування. Можливе наслідування від більшості вбудованих типів та типів розширень. Метапрограмування в мові дозволяє швидко та елегантно реалізовувати складні шаблони проектування. Python-фреймворки, такі як Django, DRF, SQLAlchemy, використовують метапрограмування для забезпечення легкої розширюваності та повторного використання коду.

Характерними для мови є чіткість та послідовність синтаксису, модульність та розширюваність. Модулі Python (розміщені в каталогах файлової системи або ZIP-архівах) об'єднуються в пакети, формуючи розроблені додатки чи бібліотеки. Для отримання доступу до простору імен модуля використовується оператор підключення import.

Репозиторій ПЗ для мови Python (PyPI) містить велику кількість сторонніх модулів:

– для веб-розробки (фреймворки, мікрофреймворки, системи управління контентом, засоби для синтактичного аналізу веб-сторінок, для підтримки різних Інтернет-протоколів та роботи з повідомленнями електронної пошти);
– для організації взаємодії з БД (пакети для доступу до різних СУБД, зокрема PostgreSQL, Microsoft SQL Server, MySQL та SQLite);
– для проведення спеціалізованих математичних та наукових розрахунків (пакети для обробки та аналізу даних та математичного моделювання, роботи з багатомірними масивами та алгоритмами оптимізації);
– для створення кросплатформних програм з GUI (пакети tkinter, wxPython; Pygame для розробки ігор та роботи з мультимедіа, а також засоби для роботи з растровою графікою).

Засоби каталогу PyPI в поєднанні зі стандартною бібліотекою Python надають розробнику будь-якого рівня необмежені можливості для розв’язання широкого класу інженерних задач.

Python може використовуватись в діалоговому режимі з дебагером та вичерпною системою довідки, що є перевагою для експериментування та розв’язання простих задач. Такий режим корисний як новачкам, так і досвідченим програмістам для тестування будь-яких фрагментів коду перед його використанням в основній програмі.

Мова програмування має вдалу і потужну реалізацію ООП, відмінну від решти об'єктно-орієнтованих мов, з наступними особливостями, крім наведених раніше:

– поліморфізм – всі функції віртуальні;
– інкапсуляція, приховані члени доступні для використання та помічені як приховані особливими іменами;

– спеціальні методи для керування життєвим циклом об'єктів: конструктори, деструктори, розподільники пам'яті;
– перевантаження всіх операторів, крім is, '.', '=' і символьних логічних;
– управління доступом до полів (частковий доступ, емуляція полів і методів);
– наявність методів для виконання поширених операцій (істинносне значення, глибоке копіювання, серіалізація, ітерація по об'єкту);
– повна інтроспекція;
– класові та статичні методи, класові поля;
– класи, вкладені у функції та інші класи.

Існує встановлений стандарт взаємодії між Python-програмами, виконуваними на стороні сервера, та самим веб-сервером (наприклад, HTTP-сервер Apache) – WSGI. Він надає простий на універсальний спосіб взаємодії між більшістю веб-серверів та веб-додатків чи фреймворків [13, 14]. WSGI визначає middleware-компоненту для надання інтерфейсів як серверу, так і веб-додатку, а також:

– обробки сесій;
– автентифікації/авторизації;
– управління URL (маршрутизації запитів);
– розподілу навантаження;
– пост-обробки вихідних даних.

Популярними сумісними з WSGI Python веб-фреймворками є Django, Flask, Dash, CherryPy, TurboGears, Pylons.

У табл. 3 узагальнено відомості про порівняні мови програмування для розробки серверної частини системи спільного управління фінансами громади.

На основі порівняльної таблиці в якості мови програмування було обрано Python – портовану мову з широким вибором доступних модулів для веб-розробки, детальною документацією, вичерпним та зручним

синтаксисом, вдалим дизайном ООП та інтерфейсами до найпоширеніших БД. Не меншою мірою вибір мови обґрунтований наявністю інтерактивних засобів тестування та налагодження коду.

Таблиця 3

Порівняльна характеристика мов програмування

Ознака	PHP	JavaScript	Ruby	Python
Мультипарадигмальність, підтримка ООП	Підтримка функціонального, процедурного стилів, слабко розвинене ООП	Підтримка функціонального, процедурного стилів, прототипне програмування на противагу ООП	Підтримка функціонального, процедурного, об'єктно-орієнтованого стилів	Підтримка функціонального, процедурного, об'єктно-орієнтованого стилів
Стандартна бібліотека	Доступна для вирішення типових проблем PHP	Відсутня	Доступна, багатофункціональні засоби	Приваблива сторона мови – велика кількість модулів
Наявність фреймворків для веб-розробки	+	+	+	+
Інтерпретованість	+	+	+	+

2.3. Вибір фреймворку для розробки веб-додатку

Побудова логіки серверної частини, користувацького інтерфейсу, навігації користувача в межах додатку з допомогою браузера зазвичай складається з повторюваних частин. Для мінімізації дублювання коду використовуються веб-фреймворки. Веб-фреймворки націлені на

реалізацію функціональних можливостей, спільних для більшості веб-додатків, таких як зіставлення URL-адрес частинам коду Python, доступ до БД, розробка інтерфейсів [15].

Для побудови масштабованого та надійного веб-додатку з простотою інтерфейсу та легкістю внесення потенційних змін від веб-фреймворку вимагається підтримка REST-архітектури. Вибір фреймворку має бути обґрунтований:

– сумісністю з WSGI;
– наявністю готових засобів шаблонізації, автентифікації, роботи з БД, зокрема підтримкою ORM;
– гнучкістю проектування структури додатку (перевага розширень бажаної функціональності над вбудованими до фреймворку інструментами);
– наявністю структурованої та вичерпної документації;
– безкоштовним розповсюдженням.

2.3.1. Веб-фреймворк Django

Django – безкоштовний фреймворк для розробки веб-додатків на мові програмування Python, що використовує шаблон проектування MVC. Типовий додаток з використанням Django являє собою структуру з кількох модульних частин, яка вирізняє веб-фреймворк з-поміж інших, наприклад Ruby on Rails. Відомий принцип фреймворку – DRY («Don't repeat yourself»).

Відмінною рисою Django вважається спосіб конфігурації обробників URL за допомогою регулярних виразів.

Для роботи з БД в Django присутнє власне ORM, в якому модель для подальшої генерації схеми БД описується за допомогою класів Python. Веб-фреймворк надає вбудовані засоби для авторизації та автентифікації, створення RESTful API, власну систему шаблонізації з тегами та наслідуванням, систему кешування, бібліотеку для роботи з формами.

Компоненти, вбудовані до ядра фреймворку, зазвичай доволі складно замінити іншими [16].

Django підтримує стандарт WSGI та може працювати під управлінням різноманітних серверів; передбачає підтримку більшості БД та власний сервер для веб-розробки.

Фреймворк має докладну документацію та є популярним рішенням в середовищі Python-розробників для написання різнопланових проектів – як інтернет-магазинів, систем управління контентом, так і вузькоспеціалізованих додатків [17]. Однак архітектура Django з її залежностями від вбудованих інструментів була створена загалом для першого типу веб-застосунків, з чого випливає її усталеність та надмірність для використання в даній роботі.

2.3.2. Веб-фреймворк Flask

Flask – фреймворк для створення веб-додатків на мові програмування Python. Flask відноситься до категорії мікрофреймворків, тому що не залежить від вбудованих засобів та бібліотек. Він побудований на основі двох компонентів. HTTP-сервер Werkzeug представляє інструментарій для WSGI-додатків, підтримує різні версії мови Python та може використовуватись для побудови веб-додатків довільної структури. Другим компонентом є Jinga2 – популярний Python-шаблонізатор, що поєднує шаблони з джерелами даних для відображення динамічних веб-сторінок [18].

Flask швидкий, поставляється безкоштовно, є простим у розгортанні та використанні, має вичерпну документацію та велику дружню спільноту розробників у мережі Інтернет.

Рисою, що відрізняє Flask від Django та інших фреймворків, є збереження ядра розроблюваного додатку якнайбільш простим, але розширюваним [19]. Таким чином, мікрофреймворк не має рівня абстракції БД, не підтримує валідацію форм, автентифікацію та інші

компоненти безпосередньо, надаючи розробнику вільний вибір серед множини розширень та доповнень для будь-якої функціональності. Це надає певну гнучкість організації коду та можливість побудови довільної структури додатку. Вищезгадані розширення часто оновлюються та достатньо легко підключаються до веб-додатку.

2.3.3. Веб-фреймворк Dash

Dash – молодий фреймворк з відкритим вихідним кодом для побудови аналітичних веб-додатків на мові програмування Python. Розроблений на підґрунті Flask, Plotly.js та React.js, Dash є зручним засобом для побудови довільних користувацьких інтерфейсів [20].

З допомогою кількох простих шаблонів фреймворк абстрагує технології і протоколи, необхідні для створення інтерактивного веб-додатку. Dash з основною функціональністю встановлюється безкоштовно з допомогою PyPI, але деякі з розширень фреймворку є комерційними.

Додатки Dash є кросплатформними та мобільними, можуть бути налаштованими як WSGI-додатки.

Засоби для автентифікації імпортуються окремим dash-auth модулем з двома методами автентифікації:

– HTTP Basic Auth, що є функціонально обмеженим;

– Plotly OAuth, що вимагає платної підписки.

Фреймворк не містить системи шаблонізації, розробник має створювати макет за допомогою Python-структур та сторонньої бібліотеки.

Спільнота розробників даного фреймворку знаходиться в процесі свого розвитку, а документація обмежена англомовною версією та, на жаль, не надає істотної інформації про роботу Dash з БД. Для отримання, додавання та видалення записів можуть використовуватись курсори БД.

У табл. 4 наводиться узагальнене порівняння розглянутих фреймворків відповідно до вимог, що випливають з поставленої задачі розробки веб-додатку для спільного управління фінансами громади.

Оптимальним вибором на основі порівняльної характеристики став розширюваний мікрофреймворк Flask.

Таблиця 4

Порівняльна характеристика веб-фреймворків

Ознака	Django	Flask	Dash
WSGI-сумісність	+	+	+
Вбудовані модулі для шаблонізації, автентифікації, роботи з БД	+	З допомогою розширень	З допомогою розширень
Гнучкість проектування структури додатку	–	+	+
Наявність детальної документації	+	+	–
Безкоштовне розповсюдження	+	+	–

2.4. Вибір СУБД

Для забезпечення швидкодії та безпеки розроблюваного додатку рекомендується обрати СУБД з клієнт-серверною архітектурою. СУБД повинна забезпечувати реляційну модель роботи зі структурованими даними та відповідати вимогам ACID, що гарантують надійне виконання транзакцій. СУБД має дозволяти одночасну модифікацію БД кількома користувачами, підтримувати ключі, збережені процедури та індекси для підвищення ефективності виконання запитів.

2.4.1. СУБД MySQL

MySQL – безкоштовна СУБД з відкритим вихідним кодом, що є поширеним рішенням для малих та середніх веб-додатків. Зазвичай використовується в якості сервера, з яким взаємодіють локальні та віддалені

клієнти, однак з дистрибутивом постачається бібліотека внутрішнього серверу, що дозволяє підключати MySQL в автономні програми.

СУБД підтримується багатьма мовами програмування, тому часто слугує для створення динамічних веб-сторінок. MySQL проста у встановленні, підтримує транзакції, кешування запитів, індексацію, повнотекстовий пошук, збережені процедури, тригери та представлення, необмежену кількість користувачів для одночасної роботи з БД [21].

MySQL продовжує свій розвиток на шляху до відповідності стандарту SQL, хоча все ще підлягає критиці за розходження з останнім, зокрема щодо трактування NULL-значень.

З найновішими оновленнями було додано можливості сегментування, реплікацію, вбудований планувальник завдань, засоби для діагностики проблем та аналізу продуктивності. Максимальний розмір таблиць може досягати 4 Гбайт.

2.4.2. СУБД PostgreSQL

PostgreSQL – потужна безкоштовна об’єктно-реляційна СУБД, що використовує та доповнює мову SQL широкою функціональністю для безпечного збереження та масштабування даних; реалізована для всіх поширених платформ та ОС.

Безперечними перевагами СУБД є надійні механізми транзакцій та реплікації, система вбудованих мов програмування, наслідування таблиць та легка розширюваність [22]. PostgreSQL надає перевірену архітектуру, цілісність даних, детальну структуровану документацію та має активну спільноту розробників [23].

PostgreSQL повністю сумісна зі стандартом ACID, підтримує велику кількість вбудованих типів даних та може бути доповнена довільними типами даних, підтримує різні рівні індексації, Multiversion Concurrency Control для кількох одночасних підключень до БД без блокування запису,

табличні простори, збережені функції та процедури, регулярні вирази та засоби генерації сумісного з іншими системами SQL-коду.

Максимальний розмір таблиці складає 32 Тбайт.

2.4.3. СУБД SQLite

SQLite – безкоштовна полегшена реляційна СУБД, реалізована у формі бібліотеки з підтримкою стандарту SQL-92. Так як SQLite не є клієнт-серверним рішенням, рушій СУБД є не окремим процесом, з яким взаємодіє застосунок, а складовою частиною програми. В якості протоколу обміну використовуються виклики функцій SQLite, що зменшує час відгуку та спрощує програму. Вся БД з даними, таблицями та індексами зберігається в єдиному файлі, на тому ПК, з якого виконується додаток [24].

Перед початком виконання транзакції файл з БД блокується. Відповідність ACID досягається створенням файлу-журналу. Кілька процесів та потоків мають одночасний доступ до даних однієї БД. Запис до БД здійснюється в тому разі, коли інші запити не обслуговуються в поточний момент часу; інакше запис не може бути виконаний. Можливий варіант автоматичного повторення спроб запису впродовж певного часу.

Механізми для роботи з СУБД розроблені та підтримуються для більшості мов програмування, з-поміж них Python. SQLite легко та без конфігурації встановлюється на поширені платформи та ОС, надає терабайтні розміри БД, легкий у використанні API, швидкість виконання найбільш типових запитів, підтримку ключів та індексів. СУБД не підтримує збережені процедури та не має зовнішніх залежностей.

Отже, перші дві СУБД, MySQL та PostgreSQL, надають клієнт-серверну архітектуру та відповідають поставленим вимогам. Вибір PostgreSQL в якості СУБД для розробки веб-додатку спільного управління фінансами громади обґрунтовується вищими показниками продуктивності, її об’єктними властивостями та всебічною підтримкою останніх стандартів SQL.

2.5. Вибір технологій для розробки клієнтської частини

При побудові веб-додатку за технологією «клієнт-сервер» клієнтська частина реалізує користувацький інтерфейс, формує запити до сервера та опрацьовує відповіді від нього. Для реалізації GUI зазвичай використовуються HTML та CSS. Формування запитів, створення інтерактивного та кросбраузерного інтерфейсу відбувається з допомогою мови JavaScript. Для організації структурованого та масштабованого коду клієнтської частини веб-додатку та зменшення його повторного використання доцільно використовувати програмні каркаси JavaScript.

На сьогодні розробникам доступний широкий вибір JavaScript-бібліотек та MV*-фреймворків для створення користувацьких інтерфейсів, які можна класифікувати наступним чином:

1. Історично важливі MVC-фреймворки (великого розміру – AngularJS, Ember; невеликого розміру - Backbone). Розділення роботи з даними та їх представленнями відбувається відповідно до шаблону проектування MVC.
2. Найбільш актуальні фреймворки (великого розміру – AngularJS, невеликого розміру – React, Vue.js). Різні за розміром та властивостями, ці рішення якісно вирізняються наявністю великих спільнот розробників та кількістю доступних навчальних матеріалів.
3. Нетипові молоді фреймворки, спільнота та можливості яких знаходяться в процесі розвитку (Reason, Polymer, Elm, Inferno) [25].

Для розробки клієнтської частини веб-додатку доцільно обрати фреймворк компактного розміру з активною спільнотою та детальною документацією.

Серед функціональних вимог до фреймворку динамічне зв’язування інтерфейсів, багаторазове використання компонентів, зокрема можливість вкладеного використання для побудови довільних інтерфейсів.

2.5.1. Angular

Angular (починаючи з 2 версії) – безкоштовний написаний на мові TypeScript MVC-фреймворк для розробки веб-додатків; має розвинуту спільноту, документацію великого об’єму та довготривалу підтримку компанії Google. Розмір фреймворку складає 563 кБайт. Ключові особливості: об'єктно-орієнтоване програмування, система типізації, двостороння синхронізація моделі та відображення, маршрутизація, шаблони на основі HTML, наявність сторонніх розширень в мережі Інтернет.

Рендеринг відбувається на стороні сервера. Фреймворк є громіздким для прототипування та має достатньо високий поріг входу.

2.5.2. React

React – безкоштовна JavaScript бібліотека для створення інтерфейсів користувача шляхом часткового оновлення вмісту веб-сторінок. Підтримується Facebook, Instagram, великою незалежною спільнотою розробників; офіційна документація добре структурована та містить практичні навчальні матеріали. Розмір фреймворку складає 100.81 кБайт. У веб-додатках React опрацьовує лише користувацький інтерфейс, що робить його швидким, простим та масштабованим. React порівнює віртуальний DOM з DOM браузера для визначення найефективнішого оновлення сторінки. Опис компонента підтримує наслідування для повторного використання компонентів. Для побудови довільних інтерфейсів компоненти можуть бути вкладеними. Рендеринг відбувається на стороні сервера.

2.5.3. Vue.js

Vue.js – JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних з допомогою реактивного зв’язування даних. Незважаючи на значну популярність в

мережі, розробка фреймворку не підтримується великими компаніями. Маючи багато спільного з React та деякі спільні риси з Angular (наприклад, директиви та шаблони), Vue.js вважається найлегшим для опанування фреймворком з найвищою продуктивністю. Добре документований, має розмір 333.46 кБайт. Особливості фреймворку: синтаксис шаблонів на основі HTML, використання віртуального DOM, реактивне програмування. Для динамічного зв'язування даних використовується директива v-bind. Міксини Vue.js дозволяють інкапсулювати частини функціональності для повторного використання в різних компонентах додатку.

Отже, було розглянуто основні особливості популярних засобів для створення користувацьких інтерфейсів. З огляду на відповідність вимогам, ефективність рендерингу, найменший об'єм та порівняно невисокий поріг входу рішенням для розробки клієнтської частини веб-додатку спільного управління фінансами громади було обрано бібліотеку React.

Для спрощення та пришвидшення процесу верстки веб-сторінок рекомендується використання CSS-фреймворку [26]. CSS-фреймворк має задовольняти наступні вимоги: підтримка кросбраузерності; підтримка RWD (адаптивного веб-дизайну сторінок); невеликий розмір фреймворку.

2.5.4. Bootstrap

Bootstrap – один з найвідоміших CSS-фреймворків на сьогодні; містить шаблони CSS та HTML для розробки дизайну веб- і мобільних додатків, типографіки, форм, кнопок, навігаційних панелей та інших компонентів інтерфейсу. Bootstrap надає підтримку більшості сучасних веб-браузерів та відповідає принципам адаптивного веб-дизайну.

Структура Bootstrap є модульною, серед основних інструментів фреймворку – сітки, шаблони, форми, таблиці, типографіка, мультимедіа, навігація, сповіщення тощо. Добре документований, доступні навчальні матеріали.

2.5.4. Semantic UI

Semantic UI – фреймворк для створення користувацьких інтерфейсів шляхом повторного використання компонентів, з семантичним підходом до побудови веб-сторінок. Semantic UI розглядає слова та класи як тотожні поняття з метою надання коду більшої читабельності та зрозумілості, використовує JavaScript та надає функціональність для налагодження коду. Фреймворк містить велику кількість готових компонентів та підтримується більшістю сучасних браузерів. Документація та додаткові ресурси поширюються здебільшого англійською мовою.

Порівняння клієнтських фреймворків також наводиться в табл. 5. За результатами аналізу обидва фреймворки так чи інакше задовольняють поставлені вимоги та містять багато доступних компонентів. Перевагу для використання в розроблюваному додатку доцільно надати Bootstrap, з огляду на менший розмір фреймворку.

Таблиця 5

Порівняльна характеристика CSS-фреймворків

Ознака	Bootstrap	Semantic UI
Підтримка веб-браузерами	Chrome, Firefox, Safari, Opera, IE, Microsoft Edge	Chrome, Firefox, Safari, Opera, IE
Підтримка RWD	+	+
Розмір фреймворку	580 кБайт	806 кБайт
Кількість колонок	12	Є можливість налаштування
Сітка	Фіксована/адаптивна	Адаптивна
Модульність структури	+	+
Доступність сторонніх плагінів	+	–

3. ХАРАКТЕРИСТИКА РОЗРОБЛЕНОЇ СИСТЕМИ

3.1. Формалізований опис вимог

Вимоги до ПЗ надають характеристику функцій та умов, у межах яких повинен працювати додаток для розв'язання задач користувача. Вимоги мають бути позбавлені прив'язки до способів реалізації ПЗ, внутрішньої архітектури додатку, за винятком зазначення цільових платформ.

Поширеним способом представлення вимог до програмної системи є створення списків вимог. Традиційність способу пояснює його використання в першому розділі даної роботи під час обґрунтування вибору критеріїв оцінювання ПЗ. Проте для повного представлення роботи розроблюваної системи, демонстрації зв'язку зібраних вимог було проведено реорганізацію документування із врахуванням рівнів та типів вимог до веб-додатку. Класифікація вимог за новим підходом схематично наведена на рис. 1.

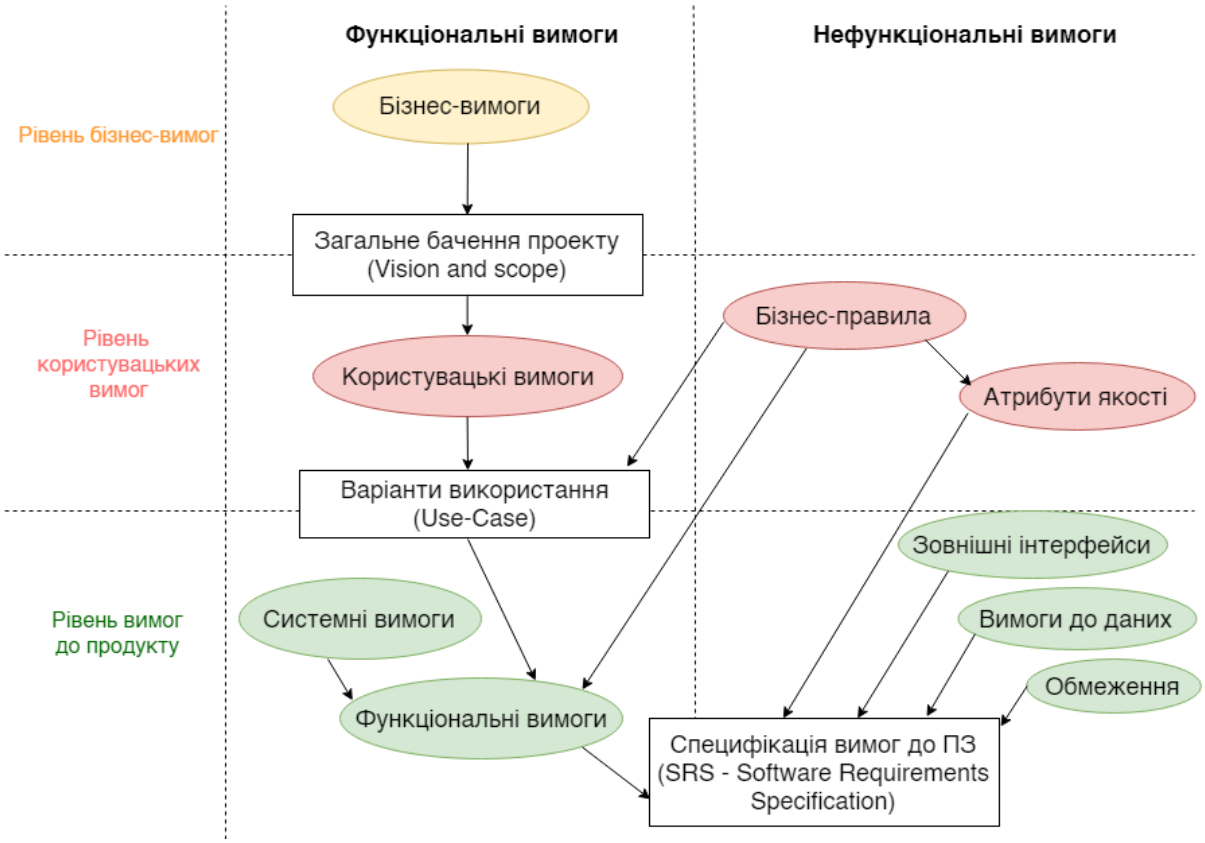


Рис. 1. Класифікація вимог із врахуванням їх рівнів та типів

Рівень бізнес-вимог: загальне бачення системи і мета розроблення

Пропонується розробка програмного продукту для прозорого управління фінансовими ресурсами колективів у закладах початкової та середньої освіти. Продукт має бути реалізований у формі веб-додатку та доступний користувачам різних ОС. Вимагається функціональність управління витратами та проектами для керівника громади на противагу операціям вкладення та висування ідей для учасників громади.

Рівень користувацьких вимог характеризується появою опису поведінки системи. На цьому рівні виділено окремі задачі, для розв’язання яких користувачі зможуть використовувати ПЗ. Користувацькі вимоги документовано за допомогою діаграми прецедентів (варіантів використання) [27].

Рівень користувацьких вимог: варіанти використання

Прецеденти в системі визначено для дійових осіб Адміністратора та Учасника.

1.1. Зафіксовані прецеденти, спільні для визначених дійових осіб:

1. Реєстрація в системі.
2. Авторизація в системі.
3. Перегляд поточного балансу громади.
4. Перегляд поточного балансу короткострокових проектів.
5. Перегляд поточного складу громади (учасників).
6. Перегляд списку короткострокових проектів за наявності.
7. Перегляд історії грошових внесків учасників громади.
8. Перегляд діаграми витрат громади.
9. Завантаження списку учасників громади.
10. Генерація річного/місячного звіту внесків та витрат громади.
11. Генерація звіту про короткострокові проекти за наявності.

1.2. Зафіксовані прецеденти для дійової особи Адміністратора:

1. Створення нової громади зі вказанням назви, міста, валюти та початкового балансу.

2. Редагування даних про громаду.
3. Додавання учасників до громади шляхом надання посилання на електронну пошту.
4. Видалення учасника зі складу громади.
5. Підтвердження грошового внеску Учасника громади.
6. Створення короткострокового проекту.
7. Редагування короткострокового проекту.
8. Видалення короткострокового проекту.
9. Закриття короткострокового проекту.
10. Підтвердження запропонованого Учасником проекту.
11. Проведення операції «Витрати» на проект із балансу громади.
12. Проведення операції «Витрати» на короткостроковий проект із балансу проекту.

1.3. Зафіксовані прецеденти для дійової особи Учасника:

1. Приєднання до громади за посиланням, отриманим електронною поштою.
2. Перегляд особистої сторінки учасника громади з інформацією про вкладення, строки внесків, фінансування короткострокових проектів за наявності, борги за наявності.
3. Пропонування короткострокового проекту на фінансування.
4. Коментування короткострокових проектів за наявності.
5. Проведення операції «Вкладення» до балансу громади.
6. Проведення операції «Вкладення» до балансу громади за іншого Учасника.
7. Проведення операції «Вкладення» в короткостроковий проект.

Побудовану UML-діаграму прецедентів розробленої системи наведено на рис. 2.

Нефункціональними користувацькими вимогами є бізнес-правила та атрибути якості.

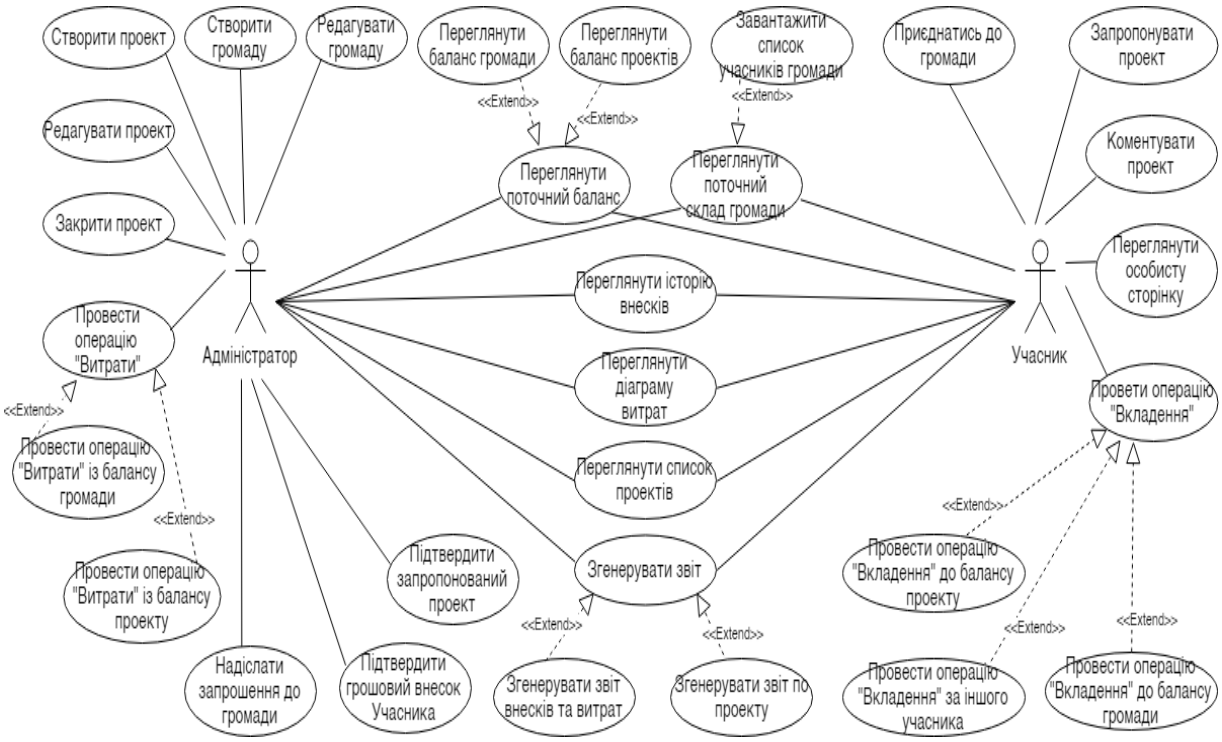


Рис. 2. Діаграма прецедентів

Бізнес-правила описують особливості та обмеження, закріплені в предметній області, що стосуються процесів і правил роботи ПЗ. Бізнес-правила розробленої системи існують в контексті її колективного використання в закладах початкової та середньої освіти та складають наступний перелік:

- фінансування запропонованого Учасником проекту можливе лише після підтвердження Адміністратором;
- функціональність системи доступна Учаснику лише в межах його громади. Способи приєднання до громади обмежені вказаними першим визначеним для Учасника варіантом використання;
- операція «Вкладення» вважається завершеною після підтвердження Адміністратором;
- операція «Витрати» вважається завершеною за умови прикріплення квитанції.

Атрибути якості на рівні користувацьких вимог доповнюють нефункціональні вимоги властивостями, відокремленими від

функціональності ПЗ, такими як продуктивність чи масштабованість [28]. Для розробленої системи суттєвою є підмножина атрибутів якості, що включає показники безпеки, модульності, надійності, зручності використання, локалізації та продуктивності. В табл. 6 наведений реєстр атрибутів якості ПЗ.

Таблиця 6

Реєстр атрибутів якості ПЗ

Код	Показник	Зміст атрибуту якості
SR1	Безпека	Паролі зареєстрованих користувачів мають зберігатися для перевірки у зашифрованому вигляді
SR2	Безпека	Захист автентифікаційних сесій користувача
SR3	Безпека	Після автентифікації користувачі мають бути захищені від CSRF-атак
SR4	Безпека	Відправлені користувачем дані мають проходити валідацію перед їх обробкою додатком
SR5	Безпека	Результати діяльності громади мають зберігатися в БД і бути доступними лише Учасникам громади
MR1	Модульність	Організація додатку має бути модульною та містити незалежні функціональні компоненти з подальшою реєстрацією їх додатком
RR1	Надійність	Самостійне відновлення роботи додатку в разі виникнення помилки
RR2	Надійність	При одночасній роботі повного складу громади з додатком найменший час між виникненням збоїв в роботі додатку має складати більше 100 годин

Продовження табл. 6

UR1	Зручність використання	Валідація користувацьких даних має супроводжуватися змістовними повідомленнями в разі некоректних даних
UR2	Зручність використання	Досягнення мети кожного варіанту використання має відбуватись не більше, ніж в 4 переходи між сторінками додатку
UR3	Зручність використання	Очевидне використання традиційних елементів інтерфейсу має забезпечити мінімальний час, потрібний для ознайомлення з додатком (60 хвилин)
LR1	Локалізація	Додаток має підтримувати українську мову
PR1	Продуктивність	Максимальний час відгуку додатку на запит користувача має не перевищувати 1 секунду
PR2	Продуктивність	Значення максимальної кількості одночасних підключень до системи має бути рівним 100
PR3	Продуктивність	Максимальний час генерації звітності має не перевищувати 2 хвилини

На рівні вимог до продукту системні, функціональні та нефункціональні вимоги формують кінцеву специфікацію ПЗ.

Обмеження регулюють способи проектування та реалізації програмного продукту, доступні розробнику. Вимоги до зовнішніх інтерфейсів встановлюють правила взаємодії системи з іншими системами, наприклад ОС. Вимоги до даних описують структури даних розроблюваного ПЗ: формати, типи даних, дозволені значення та значення за замовчуванням.

На рівні вимог до продукту було визначено обмеження, вимоги до зовнішніх інтерфейсів та даних додатку. В табл. 7 подається реєстр вимог даного рівня.

Таблиця 7

Реєстр вимог до продукту

Код	Тип вимоги	Зміст вимоги
CONST1	Обмеження: апаратні вимоги до сервера	Об’єм оперативної пам’яті від 2 Гбайт; UNIX-подібна ОС; від 8 Гбайт вільного місця на диску для збереження коду додатку; зв'язок з мережею Інтернет
CONST2	Обмеження: апаратні вимоги до клієнта	Наявність одного з сучасних веб-браузерів: Chrome, Firefox, Safari, Opera; зв'язок з мережею Інтернет; Linux, Windows або MacOS в якості встановленої ОС
EXT1	Вимоги до зовнішніх інтерфейсів	Обмін даними між клієнтською та серверною частинами додатку при виконанні фонових AJAX-запитів має відбуватись у форматі JSON
EXT2	Вимоги до зовнішніх інтерфейсів	Для надсилання запрошень до громади має використовуватись протокол SMTP
DAT1	Вимоги до даних	Дані про діяльність громади мають зберігатися в БД впродовж 2 років, дані про проектну діяльність впродовж 1 року
DAT2	Вимоги до даних	Модель представлення даних про проект має описувати категорію проекту, назву, кошторис, поточний баланс, стан, строки вкладень та реалізації

Об’єднання артефактів описаних рівнів функціональних і нефункціональних вимог являє собою повну специфікацію вимог до ПЗ, що використовується в розробці та тестуванні.

В даній роботі вимоги до ПЗ були формалізовані з метою досягнення відповідності описаній класифікації. Визначених типів вимог за трьома її

рівнями достатньо для розуміння роботи, проектування та реалізації додатку спільного управління фінансами громади.

3.2. Загальний опис системи

Згідно з вимогами розроблена система є веб-додатком модульної структури, що забезпечує функціональність для керівника та учасників у межах громади закладу початкової та середньої освіти та надає інтуїтивно зрозумілий користувацький інтерфейс.

3.2.1. Робота керівника зі вмістом системи

Роботу з системою починає керівник у ролі Адміністратора. Привітальний текст на стартовій сторінці додатку заохочує користувача зареєструватися в системі та почати роботу з громадою. Верхня навігаційна панель містить компоненти для реєстрації та автентифікації в системі, що також доступні Учаснику громади. Функціональність автентифікації в системі виділена в окремий логічний модуль. Його елементи надаються всім типам користувачів. Це забезпечує зручну для розробки організацію коду додатку і можливість його повторного використання, що є не менш важливим.

На сторінці реєстрації користувач заповнює відповідну форму, вказуючи особисті, контактні дані та пароль. Логіном користувача є електронна адреса.

При успішному проходженні перевірки достовірності введеного паролю керівник громади потрапляє на головну сторінку додатку зі вказівками щодо використання системи. Натискання на кнопку «Меню» викликає появу бічної панелі з посиланням для створення громади. Створені громади доступні керівнику на цій панелі після заповнення даних відповідної форми та застосування змін.

Посилання з бічної панелі керівник може використовувати для навігації до сторінок створених громад. Кожна така сторінка надає засоби для:

- редагування даних про громаду;
- управління складом громади;
- управління короткостроковими проектами;
- управління поточним балансом;
- генерації звітів;
- виконання адміністраторських підтверджень.

Для переходу до засобів управління керівнику доступні відповідні вкладки на сторінці громади.

На вкладці редагування громади керівник може налаштувати місто та назву.

Вкладка управління складом громади містить елементи для розширення керівником поточного складу громади шляхом запрошення Учасників:

- форму з особистими та контактними даними члена громади, серед яких обов’язковим полем є електронна пошта. Дане поле використовується для надсилання електронного листа з запрошенням та посиланням на сторінку реєстрації Учасника в громаді;
- кнопку завантаження CSV файлу з переліком електронних адрес членів громади для надсилання запрошень групі одержувачів одним запитом.

Застосування кожного зі способів запрошення Учасників до громади інформує керівника про успішність виконання запиту. В правій частині вкладки управління складом громади знаходиться список поточних Учасників, кожен елемент якого містить ім’я, збережене при реєстрації Учасника, поряд із кнопкою видалення Учасника. Для видалення від керівника вимагається підтвердження в модальному вікні.

Функціональність для роботи зі складом громади інкапсульована в логічний модуль управління Учасниками.

На вкладці управління короткостроковими проектами керівнику доступні:

- перелік поточних проектів у формі списку зліва з даними про кожен елемент;
- розширена інформація про обраний зліва проект. За замовчуванням надається інформація про перший елемент зі списку проектів;
- кнопка переходу на сторінку обраного проекту;
- кнопка для створення проекту. Викликає перехід на сторінку додавання нового проекту Адміністратором.

Окрім розширеної інформації про проект, на сторінці обраного проекту керівник може використати:

- кнопку для переходу до редагування значень наявних полів та збереження змін;
- кнопку видалення проекту для очищення даних про проект з БД;
- кнопку закриття проекту. При закритті проекту Адміністратором змінюється значення поля «Статус» проекту, а в користувацькому інтерфейсі Учасників громади блокується кнопка для проведення вкладення до проекту;
- кнопку генерації звіту по проекту, що доступна всім типам користувачів;
- кнопку для проведення витрати. Викликає перехід на сторінку проведення операції «Витрати» Адміністратором. Для здійснення операції вимагається введення суми витрачених коштів у поле «Витрачено», завантаження файлу квитанції та натискання кнопки підтвердження для зменшення поточного балансу проекту на введену суму коштів і повернення до проекту;
- коментарі Учасників до даного проекту, якщо такі наявні.

На сторінці додавання нового проекту Адміністратор вводить дані про бажаний проект у форму створення проекту. За замовчуванням значення поточного балансу новоствореного проекту є нульовим, а поле «Стан» має значення «Відкритий», що активує кнопку вкладення до проекту для Учасників громади.

Вкладка для управління поточним балансом громади призначена для:

- показу балансу громади;
- показу історії внесків та витрат у вигляді списку транзакцій з даними про обсяг коштів, дату проведення та ім’я Учасника, якщо транзакція має тип «Вкладення»;
- переходу на сторінку проведення операції «Витрати» Адміністратором. Сторінка має структуру, подібну до сторінки проведення операції «Витрати» до проекту. В цьому разі внесення інформації про витрачені кошти призводить до зменшення поточного балансу громади.

Вкладка генерації звітів пропонує елементи для завантаження списку Учасників громади, а також звіту внесків та витрат громади.

Останньою на бічній панелі керівника є вкладка виконання адміністраторських підтверджень. Її зміст поділений на дві частини. В першій частині керівник працює зі списком непідтверджених грошових вкладень. При підтвердженні вкладення Учасника за наявності зв’язку транзакції з існуючим проектом відбувається збільшення балансу проекту, а за відсутності такого зв’язку збільшується поточний баланс громади.

У другій частині вкладки керівник може підтверджувати запропоновані Учасниками короткострокові проекти. Кнопка підтвердження направляє керівника на описану вище сторінку додавання проекту, де пропонується заповнити відсутні після пропонування Учасником поля та змінити значення стану проекту. Після виконання цих дій, проект доступний на вкладці управління проектами.

Сутності, з якими працює керівник в обох частинах вкладки виконання адміністраторських повноважень, можуть бути відхилені ним. Відхилені проекти та внески не зберігаються в БД та не доступні користувачам.

3.2.2. Робота учасника громади зі вмістом системи

Робота Учасника громади з системою починається із отримання електронного листа з посиланням на сторінку реєстрації. Вказавши особисті та контактні дані, Учасник може пройти автентифікацію в системі.

Авторизований Учасник громади ознайомлюється з пояснювальною інформацією на головній сторінці та може використовувати наступні вкладки:

- особистий профіль;
- короткострокові проекти;
- поточний баланс громади;
- звіти;
- склад громади.

Учасник користується особистим профілем для відстеження власних вкладень до балансу своєї громади та проектів. Вкладка також надає:

- особисті дані Учасника, використані при реєстрації в системі;
- дані про громаду та контактні дані керівника;
- список проектів, до кінця строку яких залишається два тижні;
- список уже профінансованих Учасником проектів.

Відмінність вкладки короткострокових проектів для Учасника від даної вкладки для Адміністратора полягає в наявності кнопки для пропонування проекту замість кнопки для його створення. Таким чином, Учасник може перейти до сторінки пропонування проекту, вказати бажану інформацію у формі додавання проекту та чекати на підтвердження Адміністратором.

Запропонований проект за замовчуванням отримує значення поля «Статус» – «Не підтверджений» та нульовий поточний баланс. Проект потрапляє до списку пропонованих проектів у правій частині вкладки виконання адміністраторських підтверджень на сторінці громади Адміністратора.

Як і Адміністратор, Учасник може перейти до сторінки проекту із переліку проектів на описуваній вкладці. Для нього будуть доступними:

- кнопка для проведення операції «Вкладення», за умови відкритого статусу проекту. Учасник взаємодіє з модальним вікном для введення обсягу вкладених коштів. При цьому створюється зв'язок транзакції «Вкладення» із обраним проектом. Непідтверджене вкладення потрапляє до списку в лівій частині вкладки виконання адміністраторських підтверджень. В разі підтвердження Адміністратором збільшується поточний баланс проекту;
- кнопка генерації звіту по проекту;
- можливість коментування обраного проекту.

Вкладка поточного балансу громади для Учасника характерна зміною елементу для проведення операції «Витрати» на елемент для здійснення вкладень. Робота з вкладенням до балансу громади аналогічна даній операції для балансу проекту, але зв'язок з проектом не створюється. При підтвердженні внеску Адміністратором збільшується поточний баланс громади.

Зміст вкладки для генерації звітів не відрізняється.

Учасник використовує останню доступну вкладку для перегляду складу громади.

3.3. Архітектура системи

Обраний у другому розділі тип програмного забезпечення для розробки системи та формалізовані вимоги, наведені в третьому розділі, обґрунтовують використання клієнт-серверної технології [29].

3.3.1. Клієнт-серверна архітектура веб-додатку

Розроблений додаток містить серверну та клієнтську частини і виступає в ролі клієнта для БД. Архітектура додатку побудована з використанням концепції AJAX: замість повного перезавантаження веб-сторінки використовується динамічне довантаження потрібних даних з серверної частини. Обмін даними між клієнтською та серверною частинами системи в процесах фонового довантаження відбувається у форматі JSON. Такий підхід сприяє підвищенню продуктивності та інтерактивності додатку та задовольняє обмеження зовнішніх інтерфейсів EXT1 з рівня вимог до продукту.

3.3.2. Структура програмних засобів

Для створення серверної частини веб-додатку обраний у другому розділі дипломного проекту мікрофреймворк Flask надав три головні компоненти:

- HTTP-сервер Werkzeug (засоби маршрутизації URL-адрес, налагодження програмного коду, WSGI-сервер для локальної розробки);
- шаблонізатор Jinja2 з наслідуванням шаблонів;
- пакет Python для CLI інтеграції Click.

Вбудованих компонентів фреймворку не було достатньо для реалізації коректної роботи системи, оскільки, окрім обробки HTTP-запитів та генерації HTML-сторінок, вимагалися:

– автентифікація та авторизація користувачів;

– хешування та верифікація паролів;

– взаємодія з PostgreSQL;

– обробка даних, отриманих користувачами;

– налаштування протоколу SMTP для підтримки електронних листів.

Для реалізації цих можливостей було імпортовано Flask розширення, засоби стандартної бібліотеки мови програмування Python та сторонні бібліотеки:

– Flask-Login – для управління користувацькими сесіями (задачі авторизації, виходу з системи, запам’ятовування користувацької сесії після закриття вікна браузера);
– Flask-SQLAlchemy – обгортка для відомої ORM Python-бібліотеки (задачі відображення операцій з високорівневими сутностями на команди, застосовувані до SQL таблиць);
– Pandas – програмний засіб для обробки табличних даних (задача генерації звітів);
– smtplib – модуль для встановлення SMTP-сесії (задачі інстанціювання SMTP-клієнту з захищеним з’єднанням, автентифікації з SMTP-сервером та надсилання електронних повідомлень);
– Flask-Bcrypt – хеш-утиліта.

Клієнтська частина веб-додатку призначена для комфортної взаємодії користувача з додатком та забезпечує:

– реалізацію користувацького інтерфейсу;
– формування запитів до сервера та опрацювання отриманих відповідей;
– інтерактивні веб-сторінки з динамічною зміною наповнення, часткове оновлення веб-сторінок.

Вбудованими засобами мікрофреймворку було відділено логіку додатку від представлення за допомогою шаблонів HTML-сторінок з можливістю вставки динамічного контенту. Обробка шаблонів для показу веб-сторінок кінцевим користувачам можлива з використанням функцій рендерингу Flask, що задіюють стандартний шаблонізатор. Jinja2 дозволяє оператори управління (умовні, циклічні, блочні, присвоєння, розширення та включення) і успадкування шаблонів. Щоб не обмежуватись вбудованими

засобами для реалізації користувацького інтерфейсу було інстальовано засоби React. Компонентами React структуровано код клієнтської частини додатку.

Формування HTTP-запитів до сервера відбувається з допомогою атрибутів та методів спеціального об’єкту request, призначеного для зберігання всієї інформації, відправленої клієнтом: форм, рядків, методів та функцій-обробників запиту.

Створення та захист форм від сторонніх атак реалізовано засобами розширення Flask-WTF.

Зовнішній вигляд форм, навігаційних панелей, кнопок, модальних вікон, списків та інших елементів, призначених для роботи Адміністратора та Учасників громади з системою, було визначено відповідними блоками розширення Flask-Bootstrap. Flask-Bootstrap імпортовано для використання обраного у другому розділі дипломного проекту CSS-фреймворку.

Інтеграція з об’єктно-реляційною БД PostgreSQL для збереження результатів роботи додатку та контролю цілісності даних організована засобами розширення Flask-SQLAlchemy. В архітектурі розробленої системи це розширення представляє рівень абстракції (об’єктно-реляційне відображення) над безпосереднім двигуном СУБД PostgreSQL. Інтеграція зі Flask розширенням полегшує налаштування зв’язку з базою даних порівняно з написанням коду для роботи ORM-механізму SQLAlchemy. Таким чином, створення об’єктів у системі та операції над ними відбуваються згідно зі звичними стандартами SQL, в той час як веб-додаток оперує класами, об’єктами і методами замість таблиць. Конфігурацію бази даних на рівні абстракції визначено уніфікованим локатором двигуна PostgreSQL зі вказанням серверу обслуговування БД.

Загальна структурна схема розробленої системи зображена на рис. 3.

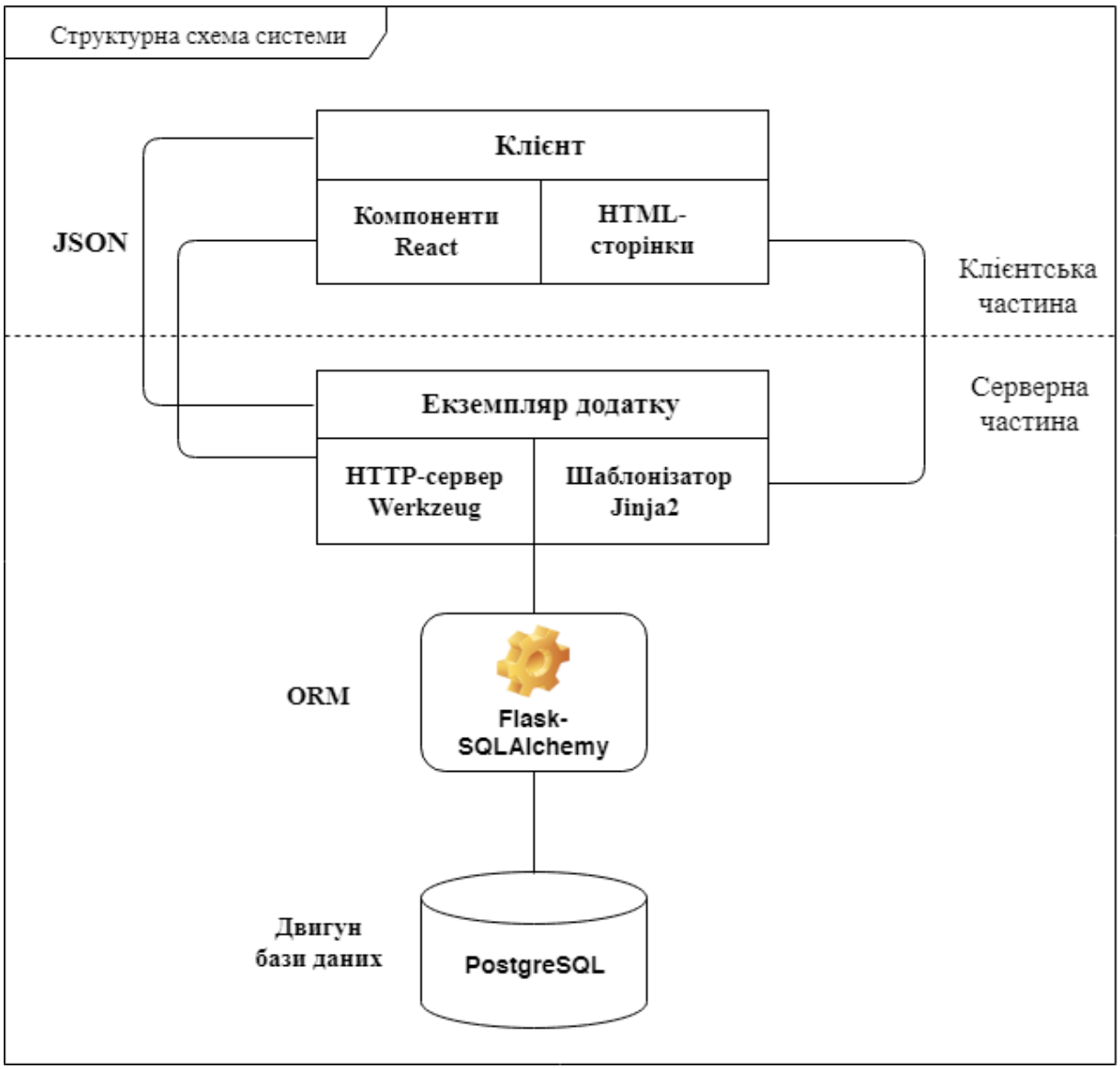


Рис. 3. Структурна схема системи

3.4. Опис структур даних додатку

При концептуальному проектуванні бази даних додатку було виділено основні сутності для опису предметної області. Наведемо текстове представлення отриманої в результаті проектування моделі даних «сутність-зв'язок» у формі переліку сутностей та їх атрибутів:

1. Користувач (User):
   – адреса електронної пошти (логін) – повинна мати унікальне значення, використовується для авторизації користувача;
   – пароль – згідно з атрибутом якості SR1 має зберігатися у зашифрованому вигляді;

– ім’я;
– прізвище;
– номер телефону – згідно з атрибутом якості SR5 має бути доступним лише користувачу;
– дата реєстрації в системі;
– статус (можливі значення: підтвердження електронної пошти, активний, видалений).

2. Тип користувача (Type):
– найменування типу (можливі значення для користувачів: адміністратор, учасник громади).

3. Громада (Group):
– назва;
– місто розташування;
– поточний баланс;
– валюта (можливі значення: UAH, USD);
– статус (можливі значення: активна, видалена);
– керівник.

4. Проект (Project):
– назва;
– категорія (можливі значення: події, подорожі, навчання, побут);
– строк вкладень – визначає дату, до якої можливе внесення коштів;
– строк реалізації – визначає дату, до якої можливі витрати коштів;
– статус (можливі значення: не підтверджений, відкритий, закритий);
– дата відкриття;
– кошторис;
– поточний баланс;
– автор.

5. Коментар до проекту (Comment):
– зміст;

– дата написання;

– автор.

6. Транзакція (Transaction):

– тип (можливі значення: вкладення, витрати);

– дата проведення;

– обсяг вкладених або витрачених коштів;

– задіяний користувач;

– цільовий проект.

Концептуальну модель «сутність-зв’язок» було перетворено у схему бази даних на основі реляційної моделі. В табл. 8 визначено відношення між сутностями системи для подальшої формалізації таблиць СУБД PostgreSQL.

Таблиця 8

Відношення між сутностями системи

Сутність 1	Сутність 2	Тип відношення
Користувач	Проект	One-to-many
Користувач	Коментар	One-to-many
Користувач	Транзакція	One-to-many
Користувач	Громада	Many-to-many
Проект	Транзакція	One-to-many
Проект	Коментар	One-to-many

Описаним сутностям відповідають таблиці User, Type, Group, Project, Transaction. Відношення між сутностями Користувача та Громади, Користувача та Типу користувача представлені допоміжними асоціативними таблицями.

На рис. 4 зображено ERD-діаграму бази даних розробленого додатку.

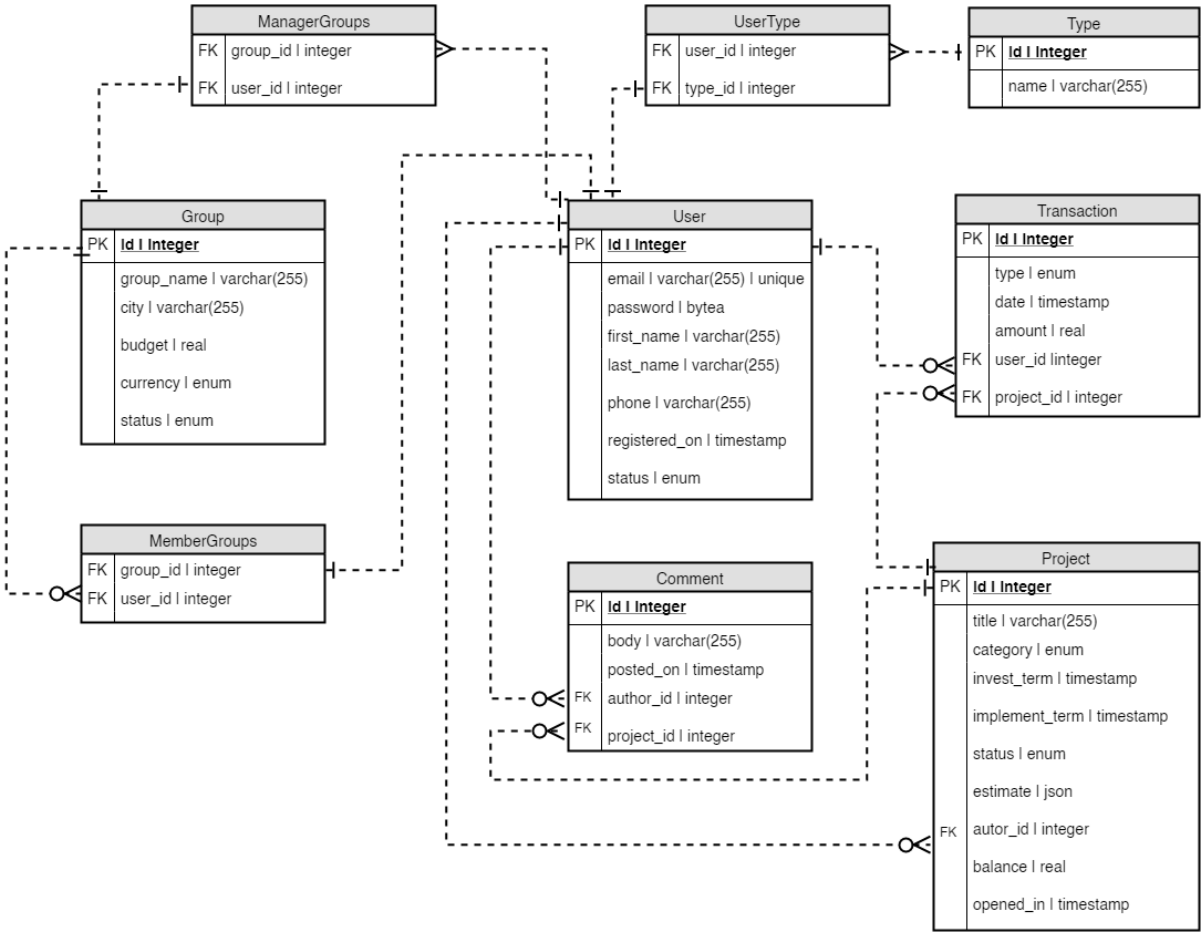


Рис. 4. Структура бази даних

4. АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Особливості реалізації

Таблиці сутностей розробленої системи, названі у третьому розділі даного дипломного проекту, представлені в програмному коді з допомогою моделей – класів мови програмування Python.

4.1.1. Моделі сутностей додатку

Поля таблиць із ERD-діаграми є атрибутами класів. Усі моделі додатку успадковуються від базового класу *db.Model*. Базовий клас забезпечує набір допоміжних класів та функцій для визначення структури моделей. Атрибути моделей є об'єктами класу *db.Column*. Кожен атрибут може приймати такі аргументи:

– тип поля таблиці БД (обов'язковий аргумент);
– параметри конфігурації атрибуту (необов'язкові аргументи, наприклад первинний ключ, індекс, унікальне значення, значення за замовчуванням, дозвіл на null-значення).

У модулі моделей розробленого додатку створено класи:

– User (додатковий аргумент поля email: унікальне значення, значення за замовчуванням поля status: email_confirmation);
– Type – набір значень типу користувачів;
– UserType – представляє тип користувача;
– Group (значення за замовчуванням поля currency: UAH, поля status: active);
– Project (значення за замовчуванням поля balance: 0.0);
– Comment – коментар користувача до проекту;
– Transaction – переказ коштів користувача;
– MemberGroups – представляє учасників громади;
– ManagerGroups – представляє керівників громад.

Зміни об’єктів створених класів є безпечними при одночасній роботі кількох користувачів з БД. Якщо під час транзакції БД (не плутати з класом Transaction) виникає помилка, зміни не будуть застосовані.

4.1.2. Модульна організація

Розміщення всіх класів моделей в окремому модулі можливе завдяки властивостям обраної мови програмування і задовольняє атрибут якості MR1 щодо модульної організації коду додатку. Основні модулі належать до пакету server:

– модуль моделей;
– модуль відправки email-повідомлень – клас EmailSender створений згідно з обмеженням EXT2 до зовнішніх інтерфейсів;
– модуль валідації даних – розроблений згідно з атрибутами якості SR4 та UR1;
– модуль імпортування учасників громади;
– модуль генерування звітності;
– модуль обробки транзакцій.

Структура додатку удосконалена з використанням концепції Blueprints – логічних компонентів для представлення функціональних частин додатку, що вміщують функції-обробники, форми, шаблони та статичні файли. У розробленому додатку реєструються наступні blueprints:

– auth_blueprint – з функціями для реєстрації, авторизації, отримання даних користувача, виходу з системи, підтвердження адреси електронної пошти;
– group_blueprint – з функціями створення громади, доступу до даних та редагування громади;
– members_blueprint – отримання та обробка списку учасників громади, функція видалення учасника із громади;

– main_blueprint – функції для предметної області додатку, зокрема додавання короткострокових проектів, написання і перегляду коментарів до проектів, перегляду користувацьких профілів.

Для інстанціювання додатку використано твірний шаблон проектування *Фабричний метод*, визначено функцію create_base_app(). Якщо розроблена система буде впроваджена в цільових установах, це рішення полегшить підхід до автоматизованого тестування [30].

Взаємодія користувача з інтерфейсом додатку забезпечена поєднанням шаблонів, засобів Bootstrap та React-компонентів. В пакеті client розміщено статичні файли та шаблони розроблених веб-сторінок.

4.1.3. Процеси перетворення даних

Схематичне зображення задіяних процесів наводиться на рис. 5.

Процес візуалізації сторінок додатку починається з запиту користувача GET (наприклад, перегляд даних про проект). За обробку запиту відповідає серверна частина. Відбір даних для подальшого представлення відбувається шляхом операцій з БД, розміщених у функціях-обробниках додатку. Обробники здійснюють рендеринг HTML-сторінок з відібраними даними та функціями клієнтської частини додатку.

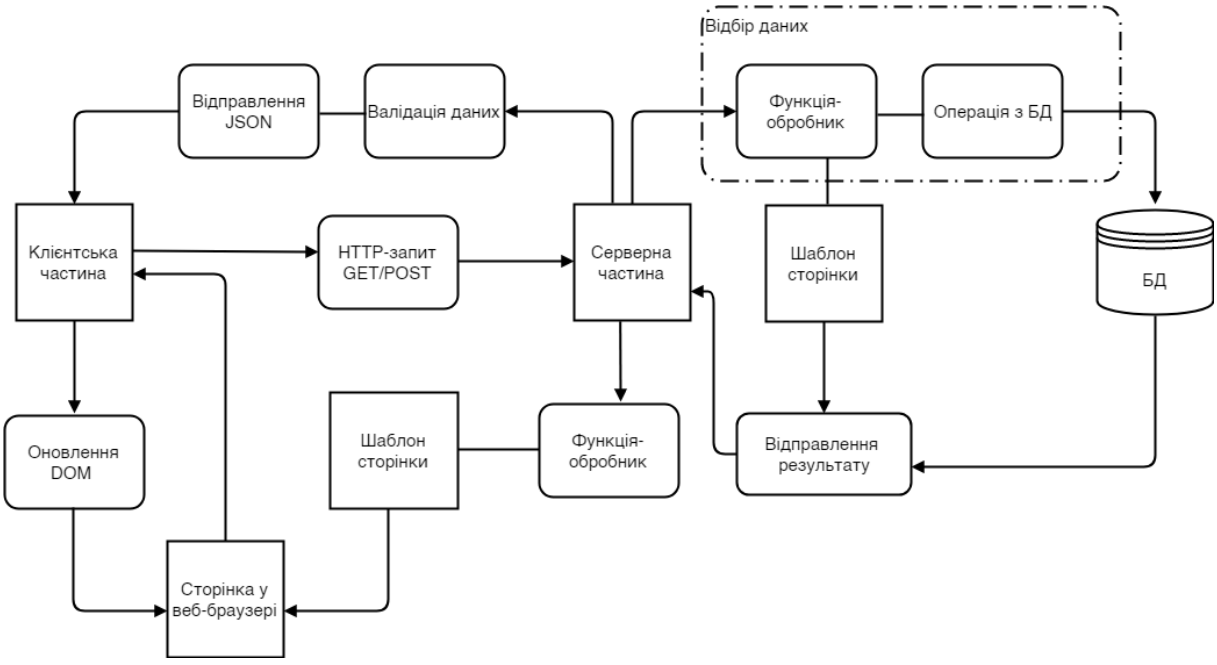


Рис. 5. Процеси перетворення даних в системі

При виконанні POST-запиту, наприклад пропонування проекту Учасником, відомості про проект проходять валідацію, надсилаються клієнтською частиною, обробляються на серверній частині, і врешті довантажуються у форматі JSON для оновлення DOM та показу на сторінці.

4.2. Тестування додатку

Згідно з принципом раннього тестування розробку тест-кейсів було розпочато після формалізації вимог до веб-додатку [31]. Користувацькі вимоги та атрибути якості стали первинним джерелом інформації для перевірки найбільш важливої функціональності. Тест-кейси, націлені на таку перевірку, зібрано в наборі для димового тестування [32].

Вид тестування: Smoke testing.

Метод тестування: Exploratory testing; розробка і виконання тест-кейсів здійснювались в один і той же час.

Набір тест-кейсів: зв’язаний.

Сценарій тестування:

1. Встановити додаток.
2. Увійти в ролі Адміністратора.
3. Створити громаду в ролі Адміністратора.
4. Запросити учасників в ролі Адміністратора.
5. Створити проект в ролі Адміністратора.
6. Зареєструватися в ролі Учасника.
7. Здійснити вкладення в ролі Учасника.
8. Запропонувати проект в ролі Учасника.
9. Підтвердити проект в ролі Адміністратора.
10. Підтвердити внесок в ролі Адміністратора.
11. Коментувати проект в ролі Учасника.
12. Видалити Учасника.

В табл. 9 подається набір тест-кейсів. Значення пріоритету не вказане з огляду на вид тестування.

Таблиця 9

Набір тест-кейсів димового тестування

№	Кроки виконання	Очікувані результати
1	1. Встановити додаток.	1. Інсталяція пройшла без помилок. Стартова сторінка додатку доступна із веб-браузеру.
2	1. Натиснути кнопку авторизації на верхній навігаційній панелі. 2. Ввести електронну адресу (логін) Адміністратора. 3. Ввести пароль Адміністратора. 4. Натиснути кнопку підтвердження.	1. Сторінка авторизації доступна із веб-браузеру. 2. У полі електронної адреси відображаються введені дані. 3. У полі паролю відображаються введені дані у вигляді зірочок. 4. Доступна головна сторінка зі вказівками про роботу системи.
3	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до створення громади. 4. Ввести назву, місто та валюту громади. 5. Натиснути кнопку підтвердження.	1. Доступна головна сторінка зі вказівками про роботу системи. 2. Доступне посилання для створення громади. 3. Доступна сторінка створення громади з формою для введення даних. 4. Введені дані відображаються у відповідних полях форми. 5. Громада доступна на бічній панелі.
4	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до сторінки громади за посиланням з бічної панелі. 4. Натиснути кнопку завантаження CSV файлу 5. Обрати файл та підтвердити дію.	1. Доступна головна сторінка. 2. Громада доступна на бічній панелі. 3. Доступна сторінка громади зі вкладкою управління складом. 4. Відкривається діалогове вікно вибору файлу. 5.З’являється інформаційне повідомлення про запрошення Учасників. На електронні адреси з вибраного файлу надіслано запрошення до громади

Продовження табл. 9

5	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до сторінки громади за посиланням з бічної панелі. 4. Натиснути кнопку створення проекту. 5. Вказати назву, категорію, строки вкладень і реалізації, кошторис. 6. Натиснути кнопку «Додати проект».	1. Доступна головна сторінка зі вказівками про роботу системи. 2. Громада доступна на бічній панелі. 3. Доступна сторінка громади зі вкладкою управління короткостроковими проектами 4. Відкривається сторінка додавання проекту з формою для введення даних. 5. Введені дані відображаються у відповідних полях форми. 6. Проект показаний у списку проектів на вкладці управління короткостроковими проектами
6	1. Відкрити клієнт електронної пошти. 2. Перейти за посиланням для вступу. 3. Ввести електронну адресу, повне ім’я, номер телефону Учасника. 4. Ввести пароль Учасника двічі. 5. Натиснути кнопку підтвердження.	1. Серед надісланих листів наявний лист з посиланням для вступу до громади. 2. У браузері відкривається сторінка реєстрації із формою для введення даних. 3. Введені дані відображаються у відповідних полях форми. 4. У полях паролю відображаються введені дані у вигляді зірочок. 5. Доступна сторінка авторизації в системі.
7	1. Увійти в ролі Учасника. 2. Обрати мишкою відкритий проект із списку та натиснути кнопку переходу до проекту. 3. Натиснути кнопку «Вкладення». 4. Ввести обсяг вкладених коштів та підтвердити дію.	1. Доступна головна сторінка із вкладкою короткострокових проектів. 2. Доступна сторінка проекту. 3. Відкривається модальне вікно для введення обсягу коштів. 4. Запис про вкладення доступний для підтвердження Адміністратором.

Продовження табл. 9

8	1. Увійти в ролі Учасника. 2. Натиснути кнопку «Запропонувати проект» на вкладці короткострокових проектів. 3. Вказати назву та категорію проекту. 4. Натиснути кнопку «Отримати підтвердження».	1. Доступна головна сторінка із вкладкою короткострокових проектів. 2. Доступна сторінка пропонування проекту з формою для введення даних. 3. Доступна кнопка «Отримати підтвердження» 4. Запис про проект доступний для підтвердження Адміністратором.
9	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до сторінки громади за посиланням з бічної панелі. 4. Обрати мишкою непідтверджений проект в правій частині вкладки. 5. Натиснути кнопку підтвердження. 6. Вказати строки вкладень і реалізації, кошторис. 7. Натиснути кнопку «Додати проект».	1. Доступна головна сторінка зі вказівками про роботу системи. 2. Громада доступна на бічній панелі. 3. Доступна сторінка громади зі вкладкою виконання адміністраторських підтверджень. 4. Доступна кнопка «Підтвердити». 5. Доступна сторінка додавання проекту з формою для введення даних. Форма частково заповнена Учасником. 6. Введені дані відображаються у відповідних полях форми. 7. Проект показаний у списку проектів на вкладці короткострокових проектів
10	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до сторінки громади за посиланням з бічної панелі. 4. Обрати мишкою непідтверджене вкладення в лівій частині вкладки. 5. Натиснути кнопку підтвердження.	1. Доступна головна сторінка зі вказівками про роботу системи. 2. Громада доступна на бічній панелі. 3. Доступна сторінка громади зі вкладкою виконання адміністраторських підтверджень. 4. Доступна кнопка «Підтвердити». 5. Баланс проекту збільшується на обсяг вкладення.

Продовження табл. 9

11	1. Увійти в ролі Учасника. 2. Обрати мишкою проект із списку та натиснути кнопку переходу до проекту. 3. Ввести текст коментаря. 4. Натиснути кнопку «Коментувати».	1. Доступна головна сторінка із вкладкою короткострокових проектів. 2. Доступна сторінка проекту із текстовим полем для введення коментарів. 3. Введений текст відображається у полі. 4. Коментар з ім’ям Учасника відображається на сторінці проекту.
12	1. Увійти в ролі Адміністратора. 2. Натиснути кнопку «Меню». 3. Перейти до сторінки громади за посиланням з бічної панелі. 4. Обрати мишкою Учасника зі списку Учасників справа. 5. Натиснути кнопку видалення Учасника. 6. Натиснути кнопку підтвердження видалення.	1. Доступна головна сторінка зі вказівками про роботу системи. 2. Громада доступна на бічній панелі. 3. Доступна сторінка громади зі вкладкою управління складом громади. 4. З’являється спливаюча підказка про можливість видалення. Доступна кнопка видалення. 5. Відкривається модальне вікно для підтвердження операції. 6. Учасник не відображається у списку справа.

З огляду на побудову додатку за технологією клієнт-серверної архітектури, було також проведено тест на з’єднання з БД.

4.3. Порівняння розробки з аналогами

Аналіз існуючих аналогів системи управління спільними фінансами громади, проведений у першому розділі даного дипломного проекту, спирався на потреби громад навчальних закладів та критерії оцінювання програмних засобів.

Вимоги до розробленої системи були складені таким чином, щоб задовольнити потреби та критерії, які не враховані існуючими на ринку рішеннями. Підсумуємо переваги:

– реалізація системи у формі веб-додатку робить її доступною з будь-якого пристрою з підключенням до мережі, незалежно від ОС;
– система не вимагає багато часу для ознайомлення та ефективного використання завдяки орієнтації на невисокий поріг входження при проектуванні користувацьких інтерфейсів;
– спільне управління коштами в системі не планується робити платною послугою;
– наявність української локалізації;
– відмова від надлишковості можливостей системи, як-то варіантів використання книги бухгалтерського обліку в AlzexFinance чи ПЗ для формування списків задач в TeamMoney;
– відмова від вузької побутової спеціалізації, як-то обмеження Acasa для контролю спільного ведення господарства;
– система передбачає проектну діяльність.

Веб-додаток з означеними перевагами може вважатися конкурентним аналогом існуючих рішень.

4.4. Рекомендації щодо подальшого вдосконалення

Розроблення веб-додатку відбувалося з намаганням підтримувати кращі практики модульної організації клієнт-серверного ПЗ. Такий підхід обґрунтовується зручністю підтримки коду, перспективою повторного використання логічних компонентів та спроможністю легко впроваджувати нову функціональність до структури вже існуючої в додатку.

Визначено такі потенційні напрямки розширення можливостей додатку:

– розробка механізму користувацьких сповіщень, наприклад, про завершення строків проектів, з метою покращення UX;

– автоматичне підтвердження вкладень у заданий час;
– надання користувачам можливості редагування непідтверджених транзакцій;
– імплементація повнотекстового пошуку проектів за назвою;
– розвантаження задач надсилання електронних листів з запрошеннями до громади, генерування звітності та списку учасників громади із серверної частини веб-додатку на сторонні робочі процеси з використанням сучасних технологій task queue;
– зміна способу автентифікації користувачів з використанням JWT-токенів для полегшення масштабування додатку, якщо виникне така потреба;
– прогнозування витрат громади на основі аналізу даних за попередні роки засобами машинного навчання.

Описані варіанти подальшої роботи представляють собою задачі різної складності та об’єму і стосуються загалом процесу розробки програмного забезпечення.

В даній дипломній роботі приділено увагу також процесу тестування програмного забезпечення. На поточний момент визначено лише стратегії мануального тестування веб-додатку. Тому рекомендується:

– вивчення концепцій фреймворку unittest для модульного тестування;
– впровадження автоматизованого тестування для перевірки повторюваних операцій в межах веб-додатку, доступності сторінок тощо.

ВИСНОВКИ

Даний дипломний проект присвячено створенню веб-додатку для спільного управління фінансами громади.

Потреби колективів у розподілі та управлінні спільними коштами було зібрано та проаналізовано. На основі потреб висунуто критерії оцінювання програмних засобів для проведення порівняльного аналізу існуючих на ринку рішень. Розглянуті аналоги не задовольняють поставленим вимогам у повній мірі, чим обґрунтовується необхідність розроблення додатку для вирішення проблем громад.

У даному проекті проаналізовано та вибрано форму розробки, програмні засоби реалізації. Доцільним визначено використання мови Python та фреймворку Flask для серверної частини; клієнт-серверної СУБД PostgreSQL; бібліотеки React та CSS-фреймворку Bootstrap для клієнтської частини додатку.

Реалізована функціональність додатку відповідає поставленим вимогам. Для початку роботи з додатком передбачена авторизація; забезпечено розділення ролей користувачів. Роль зареєстрованого користувача визначає функціональність, надану керівникам громади або учасникам: управління проектами та витратами на противагу операціям вкладення та висування потенційних проектів. Всім користувачам доступні перегляд балансу, історії внесків і генерування звітів.

У дипломному проекті розроблено архітектуру веб-додатку, структуру фінансового проекту громади та дизайн інтерфейсу динамічних веб-сторінок. Тестування додатку виконано у відповідності до затвердженої програми та методики тестування. Представлені напрямки подальшого вдосконалення додатку.

Впровадження результатів розробки сприятиме узгодженості цілей колективного фінансування, ефективності звітування та координації проектної діяльності громади.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Робота батьківського комітету [Електронний ресурс]. — Режим доступу до ресурсу: https://vseosvita.ua/library/robota-batkivskogo-komitetu-21867.html
2. Desktop Vs Mobile Vs Web Application [Електронний ресурс]. — Режим доступу до ресурсу: http://www.iomworld.com/desktop-application-vs-mobile-app-vs-web-app-2/
3. PHP – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/PHP
4. Port of node-ar-drone which allows user to control a Parrot AR Drone over PHP [Електронний ресурс]. — Режим доступу до ресурсу: https://github.com/jolicode/php-ar-drone
5. Мультипарадигмальна мова програмування – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Мультипарадигмальна_мова_програмування
6. JavaScript | MDN [Електронний ресурс]. — Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/JavaScript
7. JavaScrit – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript
8. AJAX – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX
9. Рушій JavaScrit – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Рушій_JavaScript
10. 15 самых популярных языков программирования по версии GitHub [Електронний ресурс]. — Режим доступу до ресурсу: https://habrahabr.ru/post/310262/
11. Ruby – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ruby

12. Python – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Python
13. Servers which support WSGI [Електронний ресурс]. — Режим доступу до ресурсу: https://wsgi.readthedocs.io/en/latest/servers.html
14. Frameworks that run on WSGI [Електронний ресурс]. — Режим доступу до ресурсу: https://wsgi.readthedocs.io/en/latest/frameworks.html
15. Web Frameworks – Python Wiki [Електронний ресурс]. — Режим доступу до ресурсу: https://wiki.python.org/moin/WebFrameworks
16. Django – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Django
17. Django documentation [Електронний ресурс]. — Режим доступу до ресурсу: https://docs.djangoproject.com/en/2.2/
18. Flask 1.0.2 documentation [Електронний ресурс]. — Режим доступу до ресурсу: http://flask.pocoo.org/docs/1.0/
19. Django vs Flask – 7 Amazing Comparison You Need To Know [Електронний ресурс]. — Режим доступу до ресурсу: https://www.educba.com/django-vs-flask/
20. Dash User Guide and Documentation [Електронний ресурс]. — Режим доступу до ресурсу: https://dash.plot.ly/
21. MySQL – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MySQL
22. PostgreSQL – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/PostgreSQL
23. PostgreSQL: Documentation [Електронний ресурс]. — Режим доступу до ресурсу: https://www.postgresql.org/docs/
24. SQLite – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQLite

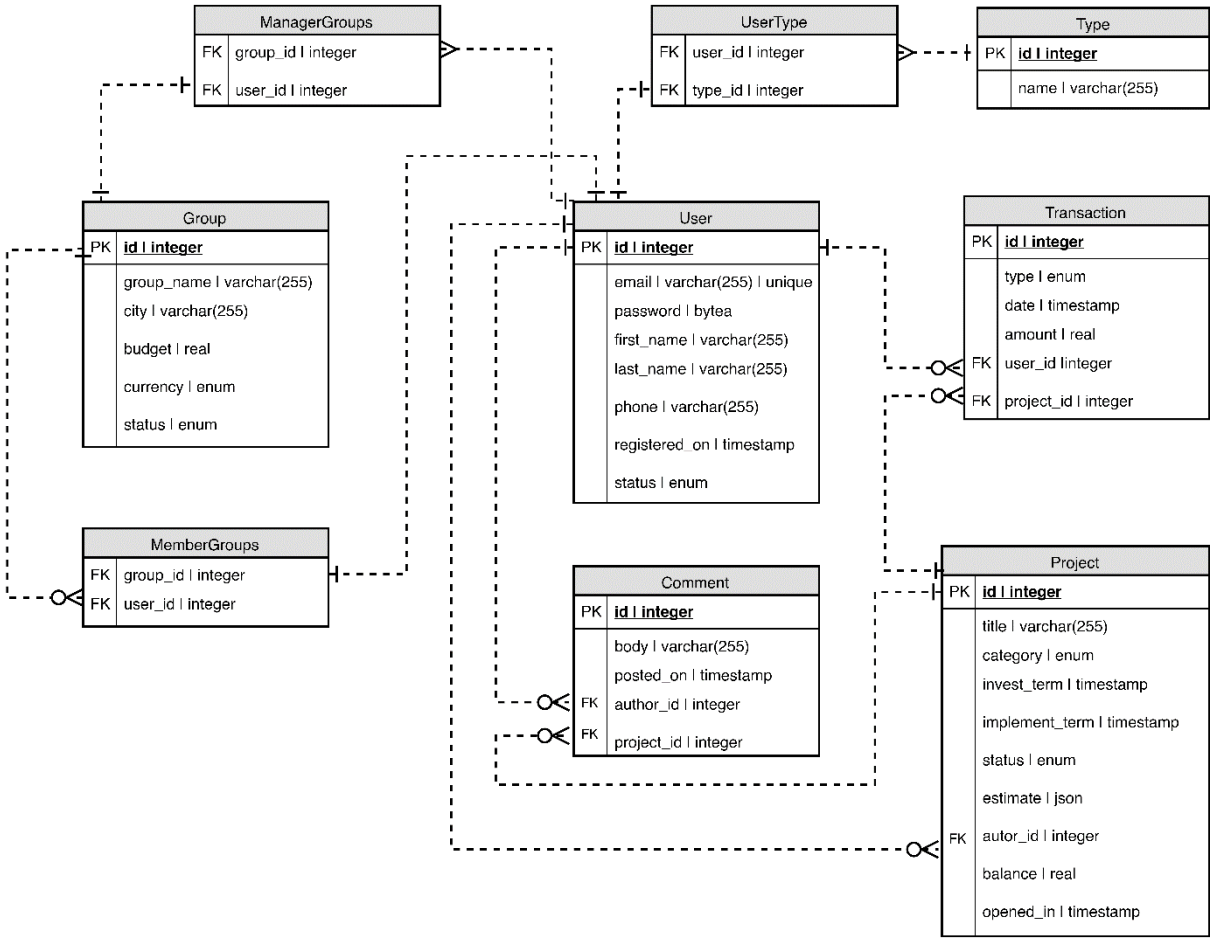
25. The Ultimate Guide to JavaScript Frameworks [Електронний ресурс]. — Режим доступу до ресурсу: https://jsreport.io/the-ultimate-guide-to-javascript-frameworks/
26. Фреймворки каскадних таблиць стилів – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Фреймворки_каскадних_таблиць_стилів
27. Сценарій використання – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Сценарій_використання
28. Нефункциональные требования к программному обеспечению. Часть 1 [Електронний ресурс]. — Режим доступу до ресурсу: https://habr.com/ru/post/231961/
29. Клієнт-серверна архітектура – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура
30. Factory method pattern – Wikipedia [Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Factory_method_pattern
31. Принципы тестирования – QALight [Електронний ресурс]. — Режим доступу до ресурсу: https://qalight.com.ua/baza-znaniy/printsipyi-testirovaniya/
32. Smoke testing – Вікіпедія [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Smoke_testing

ДОДАТКИ

Додаток 1

Копії графічних матеріалів

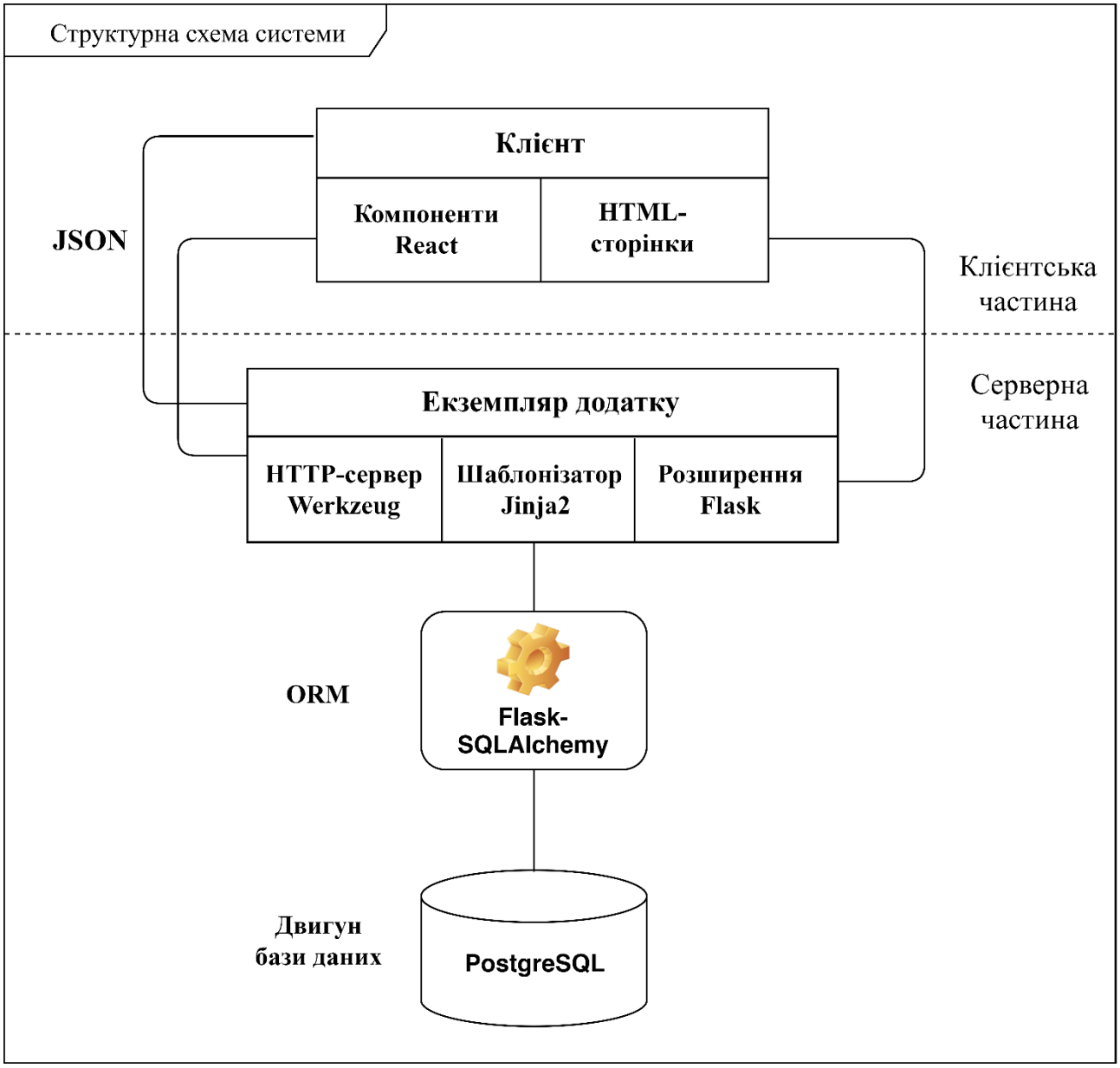
ДП.045440-06-99
Веб-додаток для спільного управління фінансами громади. Концептуальний опис поведінки системи. Діаграма прецедентів



ДП.045440-07-99

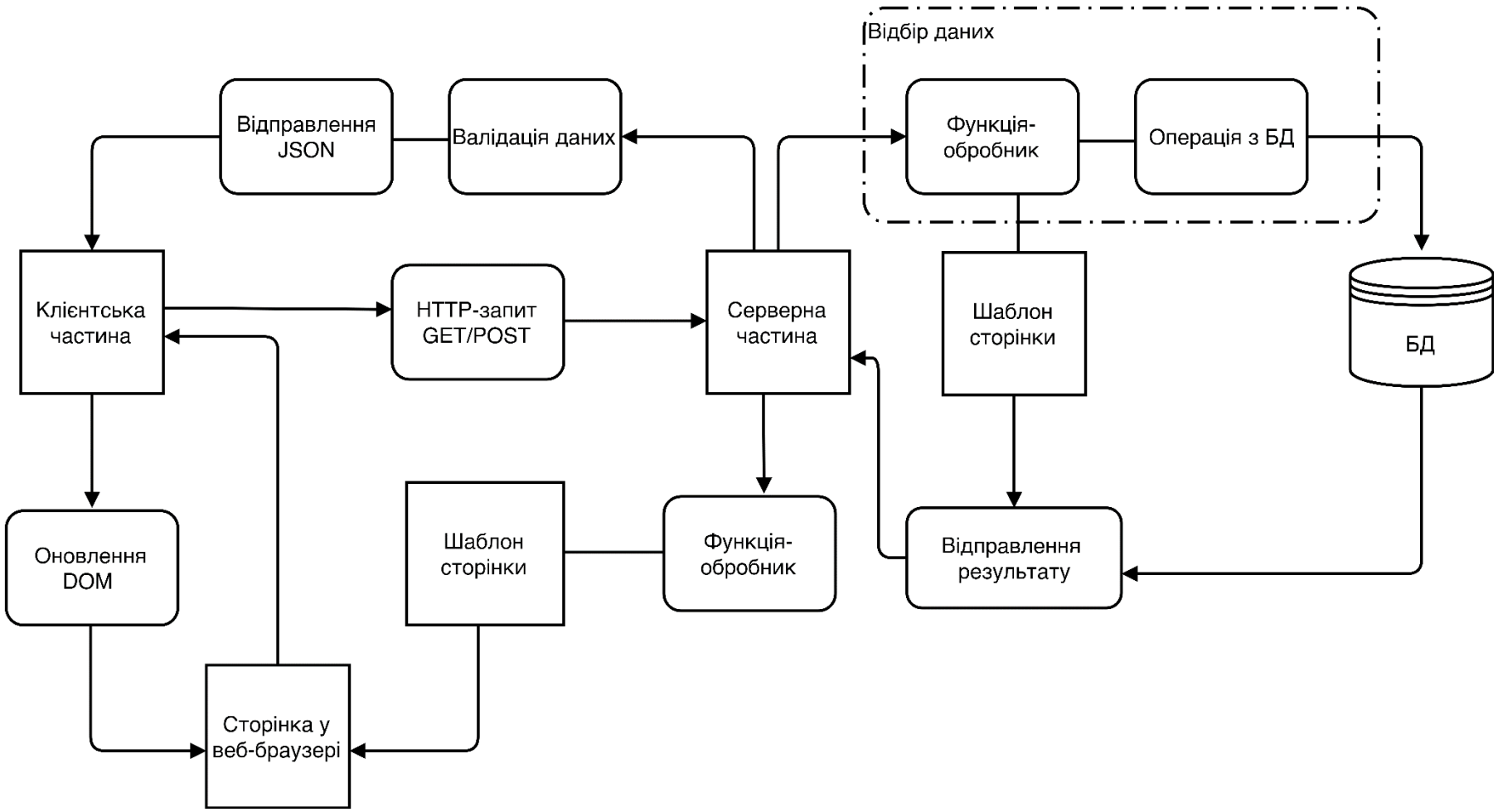
Веб-додаток для спільного управління фінансами громади. Структура бази даних. ERD діаграма

СТРУКТУРНА СХЕМА СИСТЕМИ



Ісаєва Н.М., група КП-52

ПРОЦЕСИ ПЕРЕТВОРЕННЯ ДАНИХ



Ісаєва Н.М., група КП-52

Додаток 2

Копія презентації

ВЕБ-ДОДАТОК ДЛЯ СПІЛЬНОГО УПРАВЛІННЯ ФІНАНСАМИ ГРОМАДИ

Виконала: студентка групи КП-52 Ісаєва Н. М.

Керівник: ст. викладач кафедри ПЗКС, к.т.н. Рибачок Н.А.

Київ – 2019

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити програмне забезпечення у формі веб-додатку для спільного управління фінансами громади.

1 Збір вимог

- Зібрати потреби громад навчальних закладів
- Провести огляд аналогів
- Створити технічне завдання

2 Розроблення

- Обрати засоби реалізації відповідно до вимог
- Розробити додаток

3 Тестування

- Визначити стратегію тестування
- Протестувати ПЗ

4 Планування розвитку

- Порівняти роботу з існуючими аналогами
- Окреслити напрямки подальшого вдосконалення

АКТУАЛЬНІСТЬ

Поточна ситуація: в навчальних закладах України навчається понад 4 мільйони дітей. Батьки учнів об'єднуються в громади під управлінням батьківських комітетів.

Управління фінансами та проектами залишається проблематичним аспектом взаємодії громад

Виникає потреба в інструменті для формування спільного бюджету та інформування про розподіл коштів

Розподіл коштів у навчальних закладах порівняно просто формалізувати

Сучасні батьки готові до використання нових програмних продуктів

Необхідні покращення:

- Прийняття спільних рішень про витрати
- Звітування перед колективом про зібрані та витрачені кошти
- Висування ідей для спільного фінансування
- Порозуміння між колективом та керівником щодо цілей витрачених коштів

ЗБІР ВИМОГ

Потреби громад навчальних закладів було зібрано шляхом проведення інтерв'ю з батьками учнів:

- ✔ доступ до розгорнутої інформації про фінансові проекти;
- ✔ можливість висунення власних проектів;
- ✔ можливість обговорення проектів;
- ✔ розділення прав доступу;
- ✔ невисокий поріг входження для ефективного використання ПЗ;
- ✔ підтримка всіх ОС та пристроїв.

РОЗРОБКА ВИМОГ

1. Критерії оцінювання
2. Use Case

- авторизація та розділення відповідальності
- доступ до переліку учасників громади
- перегляд поточного балансу та історії внесків
- генерування звітів за внесками та проектами

Учасник громади

- приєднання до громади
- особиста сторінка учасника
- коментування проектів
- висування власних проектів на фінансування
- внесення коштів до балансу громади та проекту

Адміністратор

- створення нової громади
- додавання учасників до громади
- видалення учасника
- створення, редагування та видалення фінансових проектів
- списання коштів на проект
- підтвердження внесків

ІСНУЮЧІ АНАЛОГИ

AlzexFinance

Десктопний додаток для ведення бухгалтерії
- генерування звітів
- платні послуги
- перевантажений інтерфейс

Team Money

Мобільний додаток для управління коштами
- транзакції
- звіти
- формування списку задач

Goodbudget

Менеджер по роботі з коштами і трекер витрат
- додавання учасників
- планування побутового бюджету
- платна підписка

ЗАСОБИ РЕАЛІЗАЦІЇ: МОВА ПРОГРАМУВАННЯ

Форма реалізації: веб-додаток ⟷

- незалежність від цільової платформи та пристроїв
- легкість встановлення, підтримки та обслуговування

Python – інтерпретована, об'єктно-орієнтована мова:

- великий об'єм стандартної бібліотеки
- множина фреймворків для веб-розробки
- детальна документація
- інтерфейси до поширених БД
- інтерактивні засоби тестування

ЗАСОБИ РЕАЛІЗАЦІЇ: ВЕБ-ФРЕЙМВОРК

Мікрофреймворк з підтримкою стандарту взаємодії WSGI:

Flask

- зберігає ядро додатку якнайбільш простим
- розширення для рівня абстракції БД, валідації форм, автентифікації користувачів
- легкість підключення та часте оновлення розширень
- гнучкість проектування структури додатку

ЗАСОБИ РЕАЛІЗАЦІЇ: СУБД

PostgreSQL – потужна об'єктно-реляційна СУБД:

- зберігання структурованих даних
- надійність виконання транзакцій
- одночасна модифікація БД кількома користувачами
- безпека та швидкодія

ЗАСОБИ РЕАЛІЗАЦІЇ: КЛІЄНТСЬКА ЧАСТИНА

Jinja

Вбудований шаблонізатор

Елементи сторінок CSS-фреймворку

Компоненти React

СКЛАДОВІ СИСТЕМИ

- ✔ Технологія: клієнт-серверна архітектура
- ✔ Формат обміну даними: JSON
- ✔ Вбудовані компоненти: HTTP-сервер, шаблонізатор

Jinja

WSGI-сервер
- локальна розробка
- маршрутизація URL-адрес
- налагодження коду

Шаблонізатор Flask
- успадкування шаблонів
- оператори управління
- генерація сторінок

СТРУКТУРА ПРОГРАМНИХ ЗАСОБІВ

Вимагалося реалізувати для коректної роботи:
- автентифікація та авторизація користувачів
- хешування та верифікація паролів
- взаємодія з PostgreSQL
- обробка даних, отриманих користувачами
- налаштування SMTP для підтримки електронних листів
- оформлення елементів сторінок

Використано розширення та бібліотеки:
- Flask-Login – управління користувацькими сесіями
- Flask-Bcrypt – хеш-утиліта
- Flask-SQLAlchemy
- pandas – задача генерації звітів
- smtplib – задача надсилання повідомлень
- Flask-Bootstrap

ІНТЕГРАЦІЯ З БАЗОЮ ДАНИХ

Flask

Python-класи, об'єкти та методи

PostgreSQL

Таблиці SQL

Flask SQLAlchemy

Об'єктно-реляційне відображення

МОДУЛЬНА ОРГАНІЗАЦІЯ ПЗ

Створення, редагування громади

Отримання списку учасників, видалення учасника

Group Blueprint

Members Blueprint

Модуль моделей

Модуль генерування звітів

Main Blueprint

Модуль авторизації

Модуль відправки повідомлень

Обробка транзакцій, додавання проектів та коментарів, перегляд профілів


РОБОТА З ГРОМАДОЮ
Створення громади
Витрати
Запрошення учасників
Модуль відправки повідомлень
Громада
Бюджет громади
Звіти
CSV
Приєднання
Вкладення
15


РОБОТА З ПРОЕКТАМИ
Закриття
Витрати
Громада
CSV
Звіти
Вкладення
Проект
Створення
Підтвердження
Пропонування
Учасники
16

ТЕСТУВАННЯ ДОДАТКУ

Мануальне тестування

- Вид тестування: Smoke testing (димове тестування)
- Метод тестування: Exploratory testing

Набір звязаних тест-кейсів

Тест на з'єднання з БД

Автоматизоване тестування

- Модульне тестування із Python unittest

- Функціональне тестування роботи у різних браузерах із Selenium

НАПРЯМКИ ВДОСКОНАЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Додаткові функції

- механізм користувацьких сповіщень про завершення строків проектів
- повнотекстовий пошук проектів за назвою
- автоматичне підтвердження вкладень у заданий час
- редагування непідтверджених транзакцій

Оптимізація роботи сервера

- розвантаження задач надсилання запрошень до громади, генерування звітності на сторонні робочі процеси

За потреби масштабування

- зміна способу автентифікації з використанням JWT-токенів

Нові задачі

- прогнозування витрат громади на основі даних за попередні роки засобами машинного навчання

ВИСНОВКИ

Виконано всі задачі процесу створення ПЗ:

1. Збору потреб та вимог
2. Розроблення додатку
3. Тестування
4. Подальшого планування

Задоволення не врахованих існуючими рішенням критеріїв:

1. Передбачена проектна діяльність
2. Безкоштовне спільне використання
3. Адаптація для різних видів не юридичних громад

Веб-додаток сприятиме узгодженості колективного фінансування, ефективності звітування та координації проектної діяльності громади.

ДЯКУЮ ЗА УВАГУ!

УНІКАЛЬНІСТЬ РОБОТИ

Частина	Унікальність, %
Вступ	98
Розділ 1	98
Розділ 2	92
Розділ 3	99
Розділ 4	98
Висновки	98
Диплом повністю	96

Додаток 3

Лістинг програми

```
from functools import update_wrapper

from flask import (
    Blueprint, redirect,
    flash, request,
    jsonify, abort
)
from flask_login import login_user, logout_user, login_required, current_user

from project.server import bcrypt, db
from project.server.model.user import User
from project.server.auth.forms import LoginForm, RegisterForm

from project.lib.user import NewUserService

auth_blueprint = Blueprint('auth', __name__, url_prefix='/auth')


@auth_blueprint.route('/register', methods=['POST'])
def register():
    form = RegisterForm(**request.get_json())
    if form.validate():
        user = User(
            email=form.email.data,
            password=form.password.data,
            first_name=form.first_name.data,
            last_name=form.last_name.data,
            phone=form.phone.data
        )
        db.session.add(user)
        db.session.commit()
        login_user(user)
        NewUserService.send_confirmation_email_msg(user=user)
        resp = {"status": "ok", "user": user.serialized()}
        return jsonify(resp)
    return jsonify({"errors": form.errors})


@auth_blueprint.route('/update', methods=['POST'])
def update():
    user: User = current_user
    form = RegisterForm(**request.get_json())
    if form.validate():
        user.first_name = form.first_name.data
        user.last_name = form.last_name.data
        user.phone = form.phone.data
        user.password = form.password.data
        db.session.add(user)
        db.session.commit()
        resp = {"status": "ok", "user": user.serialized()}
        return jsonify(resp)
    return jsonify({"errors": form.errors})


@auth_blueprint.route('/login', methods=['GET', 'POST'])
def login():
    data = request.get_json()
    form = LoginForm(**data)
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(
```

```
                user.password, data['password']):
            login_user(user)
            flash('You are logged in. Welcome!', 'success')
            resp = {"status": "ok", "user": user.serialized()}
            return jsonify(resp)
        else:
            flash('Invalid email and/or password.', 'danger')
            resp = {"status": "ok", "user": None}
            return jsonify(resp)
    return jsonify({"errors": form.errors})


@auth_blueprint.route('/user', methods=['GET'])
def get_user():
    resp = {
        "need_auth": current_user is current_user.is_authenticated,
        "user": current_user.serialized() if current_user.is_authenticated else
None
    }
    return jsonify(resp)


@auth_blueprint.route('/logout')
@login_required
def logout():
    logout_user()
    resp = {"status": "ok"}
    return jsonify(resp)


@auth_blueprint.route('/confirm_email/<int:user_id>/<token>')
def confirm_email(user_id, token):
    user = db.session.query(User).filter(
        User.id == user_id,
        User.status == User.STATUS.email_confirmation
    ).first() or abort(404)
    NewUserService.confirm_email(token=token, user=user)
    return redirect('/')

from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import DataRequired, Email, Length


class LoginForm(FlaskForm):
    email = StringField(
        'Email Address', [
            DataRequired(message="Це поле обов'язкове"),
            Email(message='Ця електронна адреса не є коректною')
        ]
    )
    password = PasswordField('Password', [DataRequired(message="Це поле
обов'язкове")])


class RegisterForm(FlaskForm):
    email = StringField(
        'Email Address',
        validators=[
            DataRequired(message="Це поле обов'язкове"),
            Email(message='Ця електронна адреса не є коректною'),
```

```
            Length(min=4, max=40, message="Електронна адреса повинна бути від 4-
ох до 40-а символів")
        ]
    )
    first_name = StringField(
        'first name',
        validators=[
            DataRequired(message="Це поле обов'язкове"),
            Length(min=4, max=40, message="Ім'я повинно бути від 4-ох до 40-а
символів")
        ]
    )
    last_name = StringField(
        'last_name',
        validators=[
            DataRequired(message="Це поле обов'язкове"),
            Length(min=2, max=40, message='Прізвище повинно бути від 2-ох до 40-
а символів')
        ]
    )
    phone = StringField(
        'phone',
        validators=[
            DataRequired(message="Це поле обов'язкове"),
            Length(min=8, max=40, message='Номер телефону повинен бути від 8-ох
до 20-а символів')
        ]
    )
    password = PasswordField(
        'password',
        validators=[DataRequired(message="Це поле обов'язкове"), Length(
            min=6,
            max=255,
            message='Пароль повинен бути від 6-ох до 255-а символів'
        )]
    )

import random

from flask import  Blueprint, request, jsonify, abort
from flask_login import current_user

from project.lib.user import NewUserService
from project.server import db
from project.server.auth.forms import RegisterForm
from project.server.model.group import Group, MemberGroups
from project.server.model.user import User
from project.tasks.import_members import import_users


members_blueprint = Blueprint('members', __name__, url_prefix='/members')


@members_blueprint.route('/import', methods=['POST'])
def import_members():
    if not current_user or 'file' not in request.files:
        abort(401)
    file_data = request.files['file'].read().decode('utf-8')
    group       =       db.session.query(Group).filter(Group.author_id       ==
current_user.id).first() or abort(404)
    import_users.delay(file_data, group.id)
```

```
    return jsonify({"status": 'ok'})


@members_blueprint.route('/create', methods=['POST'])
def create_member():
    if not current_user:
        abort(401)
    group       =       db.session.query(Group).filter(Group.author_id       ==
current_user.id).first() or abort(404)
    password = str(random.getrandbits(128))
    form = RegisterForm(password=password, **request.get_json())
    if form.validate():
        user = User(
            email=form.email.data,
            password=password,
            first_name=form.first_name.data,
            last_name=form.last_name.data,
            phone=form.phone.data
        )
        db.session.add(user)
        db.session.commit()
        member_group = MemberGroups(group_id=group.id, user_id=user.id)
        db.session.add(member_group)
        db.session.commit()
        NewUserService.send_confirmation_member_email_msg(user=user,
password=password)
        resp = {"status": "ok", "user": user.serialized()}
        return jsonify(resp)
    return jsonify({"errors": form.errors})


@members_blueprint.route('/list', methods=['GET'])
def get_list():
    members = (
        db.session.query(User)
        .select_from(Group)
        .join(MemberGroups, MemberGroups.group_id == Group.id)
        .join(User, MemberGroups.user_id == User.id)
        .filter(Group.author_id == current_user.id)
        .all()
    )
    resp = {"status": "ok", "members": [u.serialized() for u in members]}
    return jsonify(resp)

@members_blueprint.route('/delete/<id_>', methods=['GET'])
def delete(id_):
    db.session.query(MemberGroups).filter(MemberGroups.user_id == id_).delete()
    resp = {"status": "ok", "members": [u.serialized() for u in members]}
    return jsonify(resp)

from sqlalchemy import ForeignKey, Integer, Column, Table, MetaData

from project.server import db
from project.lib.types import TitledEnum, EnumInt

meta = MetaData()


class Group(db.Model):

    __tablename__ = 'group'
```

```
    class CURRENCY(TitledEnum):
        usd = 1, 'USD'
        uah = 2, 'UAH'

    class STATUS(TitledEnum):
        active = 1, 'Активный'
        deleted = 2, 'Удален'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    author_id = db.Column(db.Integer, ForeignKey('user.id'))
    # author = relationship('user')
    group_name = db.Column(db.String(255), nullable=False)
    city = db.Column(db.String(255), nullable=False)
    budget = db.Column(db.Float, nullable=False)
    currency = db.Column(EnumInt(CURRENCY), nullable=False, default=CURRENCY.uah)
    status = db.Column(EnumInt(STATUS), nullable=False, default=STATUS.active)

    def serialized(self):
        return {
            "id": self.id,
            "author_id": self.author_id,
            "group_name": self.group_name,
            "city": self.city,
            "budget": self.budget,
            "currency": self.currency.value,
            "status": self.status.value,
        }


class MemberGroups(db.Model):

    __tablename__ = 'association_member_group'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    group_id = db.Column(db.Integer, ForeignKey('group.id'))
    user_id = db.Column(db.Integer, ForeignKey('user.id'))

import datetime

from flask import current_app
from flask_login import UserMixin

from project.server import db, bcrypt
from project.lib.types import TitledEnum, EnumInt


class User(db.Model, UserMixin):

    __tablename__ = 'user'

    class STATUS(TitledEnum):
        email_confirmation = 1, 'Подтверждение емейла'
        active = 2, 'Активный'
        deleted = 3, 'Удален'

    class TYPE(TitledEnum):
        admin = 1, 'Admin'
        member = 2, 'member'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
```

```
    email = db.Column(db.String(255), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    first_name = db.Column(db.String(255), nullable=False)
    last_name = db.Column(db.String(255), nullable=False)
    phone = db.Column(db.String(255), nullable=False)
    registered_on = db.Column(db.DateTime, nullable=False)
    status          =          db.Column(EnumInt(STATUS),          nullable=False,
default=STATUS.email_confirmation)
    type = db.Column(EnumInt(TYPE), nullable=False, default=TYPE.admin)

    def __init__(self, email, password, *args, **kwargs):
        self.email = email
        self.password = bcrypt.generate_password_hash(
            password, current_app.config.get('BCRYPT_LOG_ROUNDS')
        ).decode('utf-8')
        self.registered_on = datetime.datetime.now()
        super().__init__(*args, **kwargs)

    @property
    def is_authenticated(self):
        return True

    @property
    def is_active(self):
        return self.status == self.STATUS.active

    def set_password(self, password):
        self.password = bcrypt.generate_password_hash(
            password, current_app.config.get('BCRYPT_LOG_ROUNDS')
        ).decode('utf-8')

    @property
    def is_anonymous(self):
        return False

    def get_id(self):
        return self.id

    def serialized(self):
        return {
            "id": self.id,
            "email": self.email,
            "first_name": self.first_name,
            "last_name": self.last_name,
            "phone": self.phone,
            "status": self.status.value,
            "is_admin": self.type == self.TYPE.admin
        }

    def __repr__(self):
        return '<User {0}, id:{1}>'.format(self.email, self.id)

import random


from flask import url_for, render_template

from project.server.model.user import User
from project.tasks.sender import send
from project.server import db, redis
```

```
class NewUserService(object):
    SERVER_NAME = 'http://localhost:81'

    @staticmethod
    def get_user_token(user):
        return f'USER_CONFIRM_TOKEN{user.id}'

    @classmethod
    def generate_confirfation_token(cls, user):
        token = str(random.getrandbits(128))
        redis.set(cls.get_user_token(user), token)
        return token

    @classmethod
    def send_confirmation_email_msg(cls, user):
        token = cls.generate_confirfation_token(user)
        confirmation_url   =   f"{cls.SERVER_NAME}{url_for('auth.confirm_email',
token=token, user_id=user.id)}"
        body         =         render_template('emails/email_confirmation.html',
confirmation_url=confirmation_url)
        send.delay(
            emails=(user.email,),
            subject='Підвтердження електронної адреси',
            body=body
        )

    @classmethod
    def confirm_email(cls, token, user):
        stored_token = redis.get(cls.get_user_token(user)).decode('utf-8')
        if token == stored_token:
            user.status = User.STATUS.active
            db.session.add(user)
            db.session.commit()
            return True

    @classmethod
    def send_confirmation_member_email_msg(cls, user, password):
        token = cls.generate_confirfation_token(user)
        confirmation_url     =     url_for('auth.confirm_email',     token=token,
user_id=user.id)
        body = render_template(
            'emails/email_confirmation.html',
            confirmation_url=confirmation_url,
            password=password
        )
        send.delay(
            emails=(user.email,),
            subject='Підвтердження електронної адреси',
            body=body
        )
from enum import Enum

from sqlalchemy.types import SmallInteger, TypeDecorator


class EnumInt(TypeDecorator):

    impl = SmallInteger

    def __init__(self, enum, *args, **kwargs):
```

```
        self._enum = enum
        super().__init__(*args, **kwargs)

    def process_bind_param(self, enum, dialect):
        if enum is None:
            return None
        return enum.value

    def process_result_value(self, value, dialect):
        if value is not None:
            return self._enum(value)
        return value

    def copy(self, **kw):
        return EnumInt(self._enum)


class TitledEnum(Enum):

    def __new__(cls, value, title=''):
        obj = object.__new__(cls)
        obj._value_ = value
        return obj

    def __init__(self, value, title=''):
        reserved = ('choices', 'json_choices', '_choices', '_json_choices')
        if self.name in reserved:
            raise ValueError('Illegal name: {}'.format(self.name))

        self.title = title

    @classmethod
    def choices(cls):
        """Get choices from enum
        Returns:
            list of (value, title) tuples
        """
        if not hasattr(cls, '_choices'):
            cls._choices = [(e.value, e.title) for e in cls]
        return cls._choices

    @classmethod
    def json_choices(cls, filter_func=lambda e: True):
        """Get json choices from enum
        Returns:
            list of {'value': <value>, 'title': <title>} dicts
        """
        if not hasattr(cls, '_json_choices'):
            cls._json_choices = json.dumps(
                [
                    {'value': str(e.value), 'title': e.title}
                    for e in cls if filter_func(e)]
            )
        return cls._json_choices

    @classmethod
    def as_dict(cls):
        return OrderedDict(
            [(e.name, {'value': str(e.value), 'title': e.title}) for e in cls]
        )
```

```
    @classmethod
    def get_as_enum(cls, val, default=None):
        return next((e for e in cls if e.value == val), default)

    @classmethod
    def get_as_enum_by_title(cls, title, default=None):
        return next((e for e in cls if e.title == title), default)

    @classmethod
    def get_as_enum_by_value(cls, value, default=None):
        return next((e for e in cls if e.value == value), default)
import random
from io import StringIO

from celery import task
import pandas as pd

from project.lib.user import NewUserService
from project.server import db
from project.server.model.group import MemberGroups
from project.server.model.user import User


@task(bind=True)
def import_users(self, csv_data, group_id):
    string_io = StringIO(csv_data)
    data_frame = pd.read_csv(string_io)
    for first_name, last_name, email, phone in zip(
            data_frame.first_name,
            data_frame.last_name,
            data_frame.email,
            data_frame.phone
    ):
        password = str(random.getrandbits(128))
        try:
            user = User(
                email=email,
                password=password,
                first_name=first_name,
                last_name=last_name,
                phone=phone,
                type=User.TYPE.member
            )
            db.session.add(user)
            db.session.commit()
            group_member = MemberGroups(
                user_id=user.id,
                group_id=group_id
            )
            db.session.add(group_member)
            db.session.commit()
            NewUserService.send_confirmation_member_email_msg(user=user,
password=password)
        except Exception as e:
            raise e
import importlib

from celery.app.task import Task
import celery

from project.lib.email_sender import EmailSender
```

```
def register_tasks(real_app):
    module_names = (
        'sender',
        'import_members',
    )
    for module_name in module_names:
        mod = importlib.import_module(__name__ + '.' + module_name)
        for name in dir(mod):
            attr = getattr(mod, name)
            if isinstance(attr, Task):
                attr.app = real_app


def init_celery(app):
    celery_app                  =                 celery.Celery(app.import_name,
broker=app.config['CELERY']['broker'])
    app.config.update(SERVER_NAME='localhost:81/api',
SESSION_COOKIE_DOMAIN='.localhost:81')
    celery_app.conf.update(app.config['CELERY'])
    register_tasks(celery_app)
    sender = EmailSender()
    celery_app.sender = sender
    celery_app.finalize()
    return celery_app
import unittest
import os

from flask import current_app
from flask_testing import TestCase

from project.server import create_app

app = create_app()


class TestDevelopmentConfig(TestCase):

    def create_app(self):
        app.config.from_object('project.server.config.DevelopmentConfig')
        return app

    def test_app_is_development(self):
        self.assertFalse(current_app.config['TESTING'])
        self.assertTrue(app.config['WTF_CSRF_ENABLED'] is False)
        self.assertTrue(app.config['DEBUG_TB_ENABLED'] is True)
        self.assertFalse(current_app is None)


class TestTestingConfig(TestCase):

    def create_app(self):
        app.config.from_object('project.server.config.TestingConfig')
        return app

    def test_app_is_testing(self):
        self.assertTrue(current_app.config['TESTING'])
        self.assertTrue(app.config['BCRYPT_LOG_ROUNDS'] == 4)
        self.assertTrue(app.config['WTF_CSRF_ENABLED'] is False)
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2018 р.

ВЕБ-ДОДАТОК ДЛЯ СПІЛЬНОГО УПРАВЛІННЯ ФІНАНСАМИ ГРОМАДИ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Н.А. Рибачок

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Н.М. Ісаєва

2018

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Система для спільного управління фінансами громади надає можливість відслідковування вкладених та витрачених коштів колективу та зберігає результати фінансової діяльності громади. Програмний засіб виконаний у формі веб-додатку на мові програмування Python з використанням мікрофреймворку Flask.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування в програмній інженерії перевірці підлягають: створений продукт на рівні системного тестування, формалізовані вимоги, документація, програмний код та архітектура, а також самі тести. Під час тестування додатку, що є об’єктом випробувань, має бути перевірено:

- відповідність роботи елементів веб-сторінок їх призначенню;
- наявність доступу до бази даних з результатами діяльності громади та цілісність даних при її одночасній модифікації кількома користувачами;
- коректність опрацювання користувацьких запитів;
- коректність роботи імпортованих розширень та сторонніх бібліотек;
- коректність роботи модулів: відправки email-повідомлень, валідації даних, імпортування учасників та обробки транзакцій;
- забезпечення належного рівня безпеки при роботі з системою;
- зручність роботи з інтерфейсом, швидкість ознайомлення та використання;
- відповідність функціональним та додатковим вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Для тестування було обрано метод Gray Box Testing, що являє собою комбінацію White Box Testing та Black Box Testing. Було перевірено як код, так і функціональність програмного продукту та її відповідність поставленим вимогам. Розробка та виконання тестів проводилися одночасно – за методом Exploratory testing. Тестування проведено на інтеграційному та системному рівні.

- Мета інтеграційного тестування – перевірка взаємодії між структурними частинами додатку, реєстрації логічних blueprint-компонентів.
- Мета системного тестування – перевірка взаємодії додатку з СУБД, інтерфейсу та роботи додатку як цілісної системи, побудованої з частин, що перевірені на інтеграційному рівні.

Для тестування використовуються наступні методи:

- Функціональне тестування для перевірки коректності реалізації вимог;
- Нефункціональне тестування для перевірки атрибутів якості системи;
- Мануальне тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Використано мануальний підхід до тестування. В процесі тестування підтримувалась орієнтація на загальні принципи тестування в програмній інженерії.

Розроблений веб-додаток був протестований шляхом:

1. Динамічного мануального тестування – перевірка полів, які можна редагувати, після запуску додатку.
2. Динамічного мануального тестування – перевірка загальної відповідності функціональним вимогам.

3. Тестування безпеки – автентифікаційних сесій, зберігання користувацьких паролів, захисту від веб-атак.
4. Статичне тестування коду – без запуску додатку.
5. Тестування роботи веб-додатку в різних веб-браузерах, вказаних вимогами.
6. Тестування продуктивності – перевірка часу відгуку додатку на запити користувачів.
7. Тестування зручності використання інтерфейсу.
8. Baseline-тестування – перевірка документації та вимог до веб-додатку.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ СПІЛЬНОГО УПРАВЛІННЯ ФІНАНСАМИ ГРОМАДИ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Н.А. Рибачок

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Н.М. Ісаєва

2019

ЗМІСТ

1. Опис структури веб-додатку

Розроблена система є україномовним веб-додатком модульної структури, складається зі статичних та динамічних веб-сторінок та надає функції для керівника та учасників у межах громади закладів початкової та середньої освіти.

До статичних сторінок додатку належать:

- сторінки реєстрації;
- сторінка авторизації;
- стартова сторінка Адміністратора.

До динамічних сторінок додатку належать:

- головна сторінка для Адміністратора зі вказівками щодо використання;
- сторінки створених громад;
- головна сторінка для Учасника;
- сторінка додавання проекту;
- сторінка пропонування проекту.

2. Опис вмісту веб-сторінок для керівника

Роботу з системою починає керівник громади в ролі Адміністратора. Стартова сторінка містить привітальний текст для заохочення до реєстрації в системі та початку роботи з громадою.

Верхня навігаційна панель стартової сторінки містить компоненти для реєстрації та автентифікації в системі (рис. 1), також доступні учасникам громади.

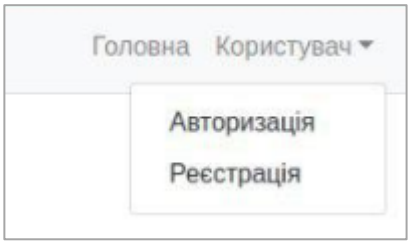


Рис. 1. Компоненти для входу до системи

На сторінці реєстрації керівник заповнює відповідну форму, вказуючи особисті, контактні дані та пароль (рис. 2). Логіном користувача є електронна адреса.

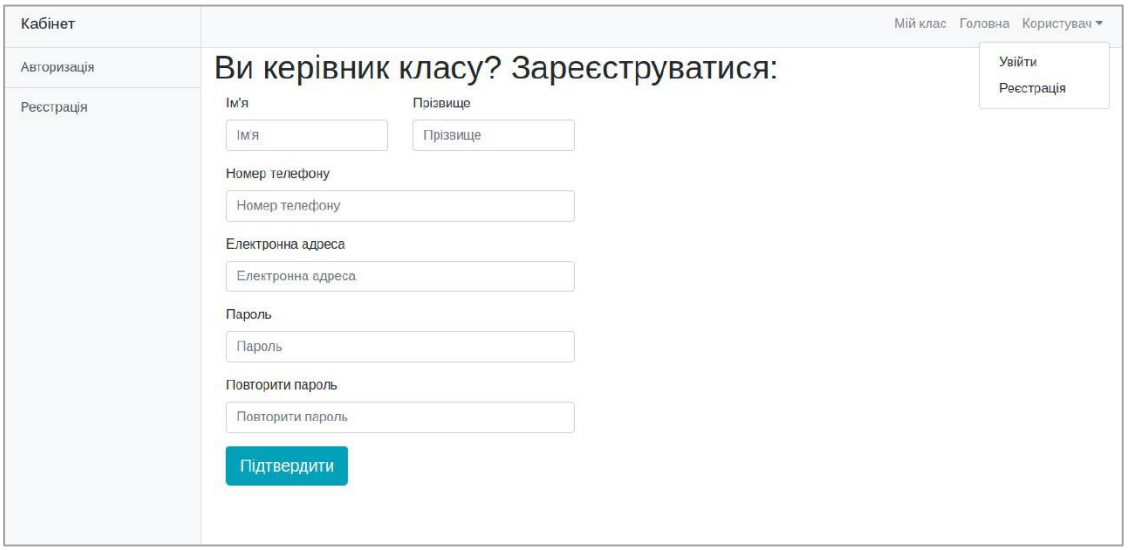


Рис. 2. Сторінка реєстрації керівника громади

За умови успішного проходження перевірки достовірності введеного паролю при подальшій авторизації в системі керівник громади потрапляє на головну сторінку додатку зі вказівками щодо використання системи. Натискання на кнопку «Меню» викликає появу бічної панелі з посиланням для створення громади. Створені громади доступні керівнику на цій панелі після заповнення даних відповідної форми та застосування змін.

Посилання з бічної панелі керівник може використовувати для навігації до сторінок створених громад. Кожна така сторінка надає засоби для:

- редагування даних про громаду;
- управління складом громади;
- управління короткостроковими проектами;
- управління поточним балансом;
- генерації звітів;
- виконання адміністраторських підтверджень.

Для переходу до засобів управління керівнику доступні відповідні вкладки на сторінці громади (рис. 3).

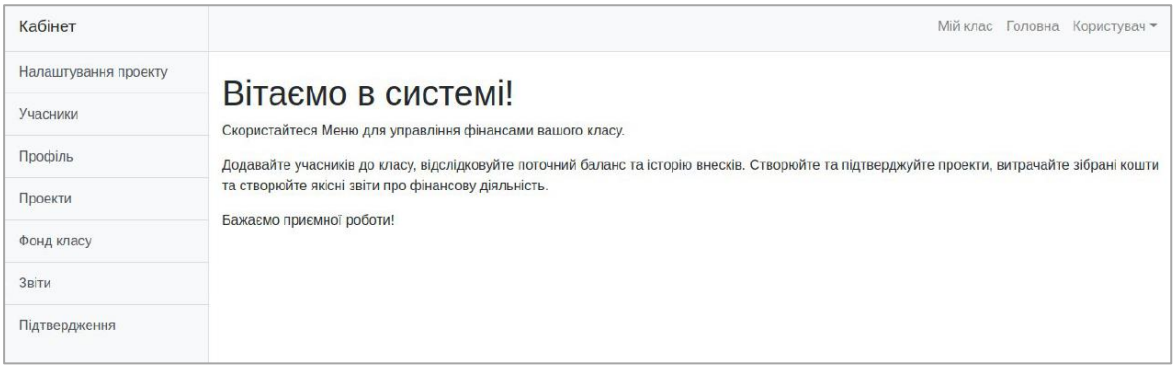


Рис. 3. Бічна панель на сторінці громади

На вкладці редагування громади (рис. 4) керівник може налаштувати місто та назву.

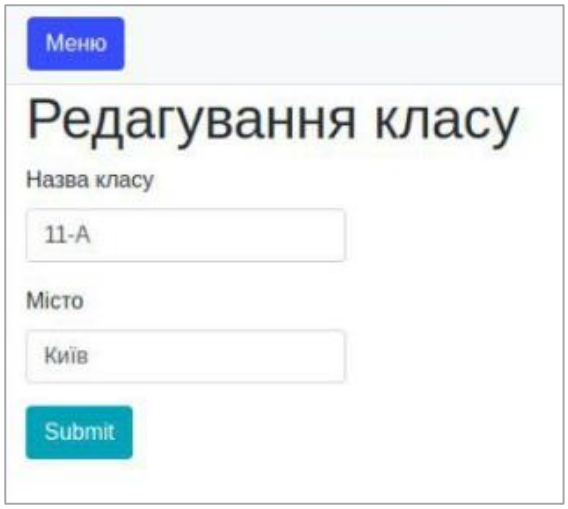


Рис. 4. Вкладка редагування громади

Вкладка управління складом громади (рис. 5) містить елементи для розширення керівником поточного складу громади шляхом запрошення Учасників:

- форму з особистими та контактними даними члена громади, серед яких обов’язковим полем є електронна пошта. Дане поле використовується для надсилання електронного листа з запрошенням та посиланням на сторінку реєстрації Учасника в громаді;
- кнопку завантаження CSV файлу з переліком електронних адрес членів громади для надсилання запрошень групі одержувачів одним запитом.

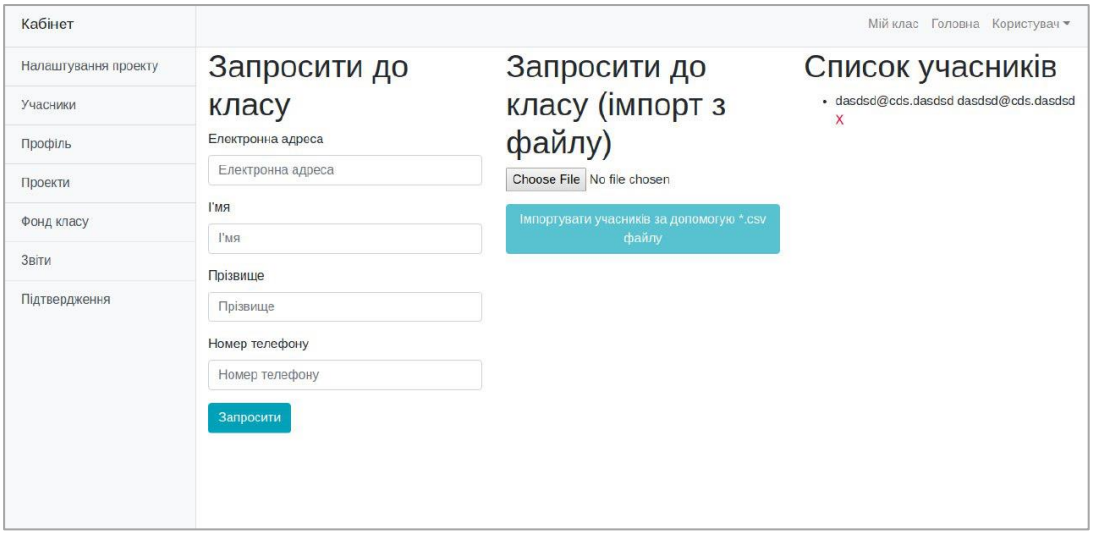


Рис. 5. Вкладка управління складом громади

В правій частині вкладки управління складом громади знаходиться список поточних Учасників, кожен елемент якого містить ім’я, збережене при реєстрації Учасника, поряд із кнопкою видалення Учасника. Для видалення від керівника вимагається підтвердження в модальному вікні.

На вкладці управління короткостроковими проектами керівнику доступні:

- перелік поточних проектів у формі списку зліва з даними про кожен елемент;
- розширена інформація про обраний зліва проект. За замовчуванням надається інформація про перший елемент зі списку проектів;
- кнопка переходу на сторінку обраного проекту;
- кнопка для створення проекту. Викликає перехід на сторінку додавання нового проекту Адміністратором.

Окрім розширеної інформації про проект, на сторінці обраного проекту керівник може використати:

- кнопку для переходу до редагування значень наявних полів та збереження змін;
- кнопку видалення проекту;
- кнопку закриття проекту. При закритті проекту Адміністратором змінюється значення поля «Статус» проекту, а в користувацькому

інтерфейсі Учасників громади блокується кнопка для проведення вкладення до проекту;

- кнопку генерації звіту по проекту, що доступна всім типам користувачів;
- кнопку для проведення витрати. Викликає перехід на сторінку проведення операції «Витрати». Для здійснення операції вимагається введення суми витрачених коштів у поле «Витрачено», завантаження файлу квитанції та натискання кнопки підтвердження для зменшення поточного балансу проекту на введену суму коштів і повернення до проекту;
- коментарі Учасників до даного проекту за наявності.

На сторінці додавання нового проекту Адміністратор вводить дані про бажаний проект у форму створення проекту. За замовчуванням значення поточного балансу новоствореного проекту є нульовим, а поле «Стан» має значення «Відкритий», що активує кнопку вкладення до проекту для всіх Учасників громади.

Вкладка для управління поточним балансом громади призначена для:

- показу балансу громади;
- показу історії внесків та витрат у вигляді списку транзакцій з даними про обсяг коштів, дату проведення та ім’я Учасника, якщо транзакція має тип «Вкладення»;
- переходу на сторінку проведення операції «Витрати» Адміністратором. Сторінка має структуру, подібну до сторінки проведення операції «Витрати» до проекту. Внесення інформації про витрачені кошти призводить до зменшення поточного балансу громади.

Вкладка генерації звітів містить елементи для завантаження списку Учасників громади, а також звіту внесків та витрат громади.

Останньою на бічній панелі керівника є вкладка виконання адміністраторських підтверджень. Її зміст поділений на дві частини.

В першій частині керівник працює зі списком непідтверджених грошових вкладень.

У другій частині вкладки керівник може підтверджувати запропоновані Учасниками короткострокові проекти. Кнопка підтвердження направляє керівника на сторінку додавання проекту, де пропонується заповнити відсутні після пропонування Учасником поля та змінити значення стану проекту. Після виконання цих дій, проект доступний на вкладці управління проектами.

3. Процедура авторизації користувачів

Авторизація користувача відбувається на сторінці авторизації. Користувач має ввести своє ім’я, електронну адресу, що є логіном, пароль та натиснути кнопку «Підтвердити», після чого користувачу доступна відповідна сторінка в залежності від його ролі.

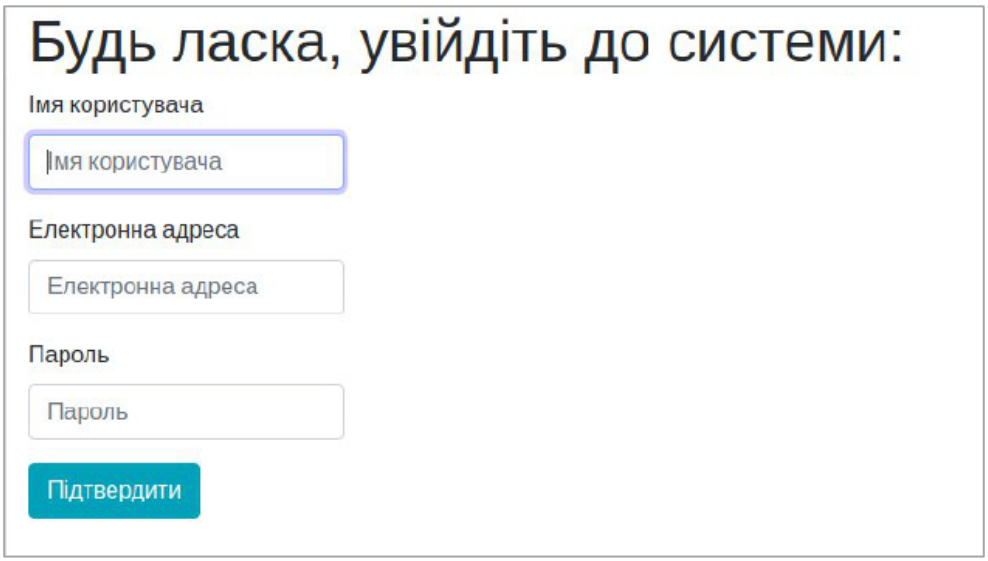


Рис. 6. Сторінка авторизації

4. Опис вмісту веб-сторінок для учасника

Робота Учасника громади з системою починається із отримання електронного листа з посиланням на сторінку реєстрації (рис. 7).

Для авторизації в системі учасник громади використовує сторінку авторизації.

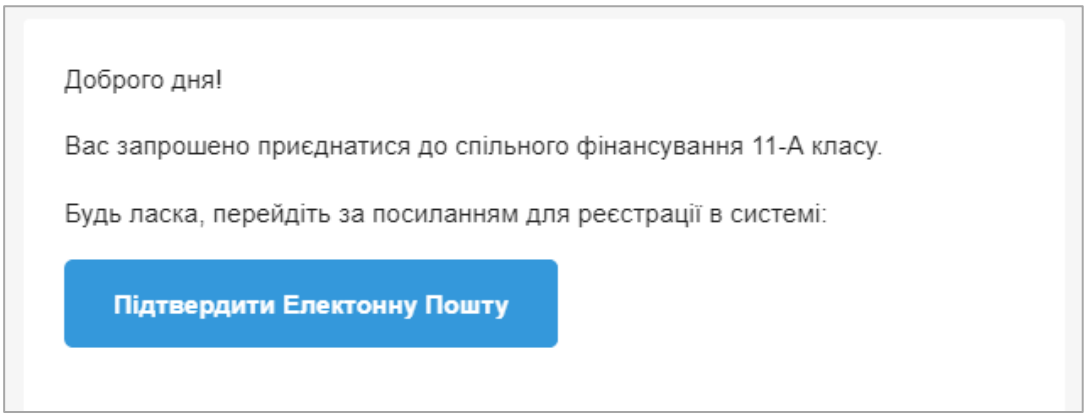


Рис. 7. Електронний лист із запрошенням до громади

Авторизований Учасник громади ознайомлюється з пояснювальною інформацією на головній сторінці та може використовувати наступні вкладки:

- особистий профіль;
- короткострокові проекти;
- поточний баланс громади;
- звіти;
- склад громади.

Учасник користується особистим профілем для відстеження власних вкладень до балансу своєї громади та проектів. Вкладка також надає:

- особисті дані Учасника, використані при реєстрації в системі;
- дані про громаду та контактні дані керівника;
- список проектів, до кінця строку яких залишається два тижні;
- список уже профінансованих Учасником проектів.

Вкладка короткострокових проектів надає Учаснику функції для пропонування проекту. Учасник може скористатися ними з допомогою кнопки пропонування. Таким чином, Учасник переходить до сторінки пропонування проекту, де може вказати бажану інформацію у формі додавання проекту та чекати на підтвердження Адміністратором.

Запропонований проект потрапляє до списку пропонованих проектів у правій частині вкладки виконання адміністраторських підтверджень на сторінці громади Адміністратора.

Учасник може перейти до сторінки проекту із переліку проектів на описуваній вкладці. Для нього будуть доступними:

- кнопка для проведення операції «Вкладення», за умови відкритого статусу проекту. Учасник взаємодіє з модальним вікном для введення обсягу вкладених коштів. Непідтверджене вкладення потрапляє до списку в лівій частині вкладки виконання адміністраторських підтверджень. В разі підтвердження Адміністратором збільшується поточний баланс проекту;
- кнопка генерації звіту по проекту;
- поле для додавання коментарів до обраного проекту.

На вкладці поточного балансу громади Учаснику не доступний елемент для проведення операції «Витрати», натомість доступним є елемент для здійснення вкладень. При підтвердженні внеску Адміністратором збільшується поточний баланс громади.

Учасник використовує останню доступну вкладку для перегляду складу громади.